EARTH PEOPLE TECHNOLOGY, Inc

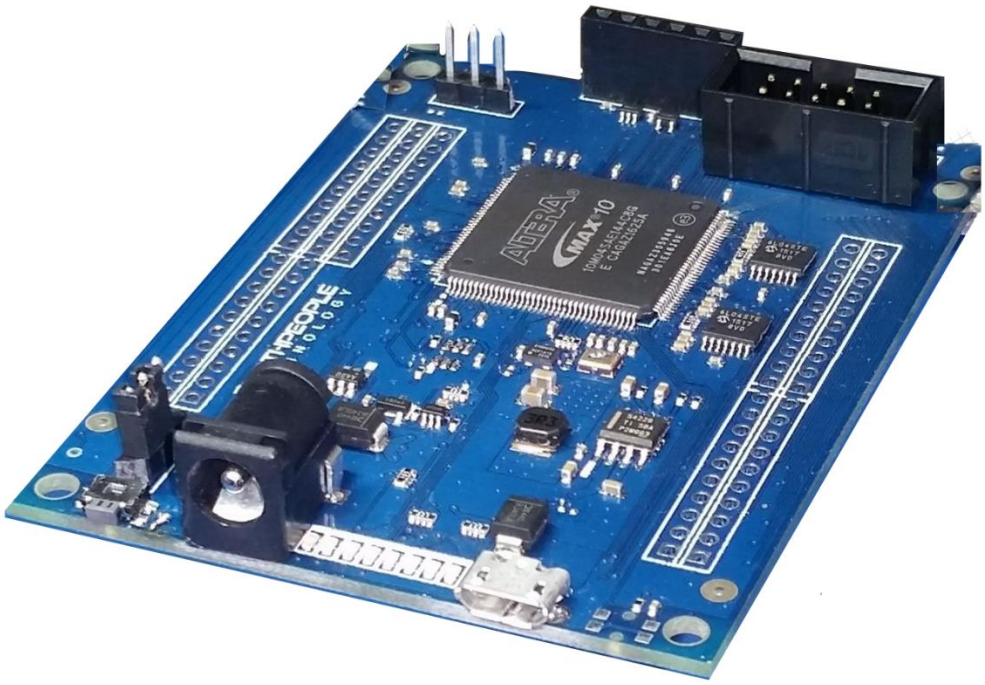
MAXPROLOGIC DEVELOPMENT SYSTEM
User Manual

The MaxProLogic is an FPGA development board that is designed to be user friendly and a great introduction into digital design for Electrical Engineering students and hobbyists. This boards provides innovative method of developing and debugging programmable logic code. It has been designed from the ground up to provide the functionality needed for the demanding projects from todays students and hobbyists. The board provides a convenient, user-friendly work flow by connecting seamlessly with Altera's Quartus II software. The user will develop the code in the Quartus environment on a Windows Personal Computer. The programmable logic code is loaded into the FPGA using only the Quartus Programmer tool.

The core of the MaxProLogic is the Intel/Altera MAX10 FPGA. This powerful chip has 4,000 Logic Elements and 200 Kbits of Memory. The MAX10 is easily scalable from the entry level college student to the most advanced projects like an audio sound meter with FFT. The MAX10 is Intel/Altera's newest entrance into low cost FPGA's.

http://www.earthpeopletechnology.com/

Contents



1 Introduction

The MaxProLogic is Open Source Hardware based on the MAX10 FPGA. Core of the board is the MAX10 chip. It has a built in Flash for configuration and incorporates 8 channels of Analog to Digital Conversion. The board has two power options, standard USB Micro B connector and 5.5mm Barrel connector. The MaxProLogic can be powered from a laptop with 2.5W of power. Or it can be run it from the +5V @ 2A wall USB chargers for 10W of power. The barrel connector can handle up to +9V @ 3 A for 27W of power. The MaxProLogic has a MicroSD connector on the bottom of the board. The board has an optional On/Off pushbutton switch that allows the user to turn the system on and off. There are two clocking options, 50MHz oscillator and 32.768KHz oscillator.

2 User Setup

The MaxProLogic is ready to go straight out of the box. Just connect power from either a PC/Laptop, +5 VDC Universal Wall Charger, +5 to 9 VDC through barrel connector, or +5 to 9VDC through pin x on Jx. Install the On/Off jumper to select default mode or On/Off switch.

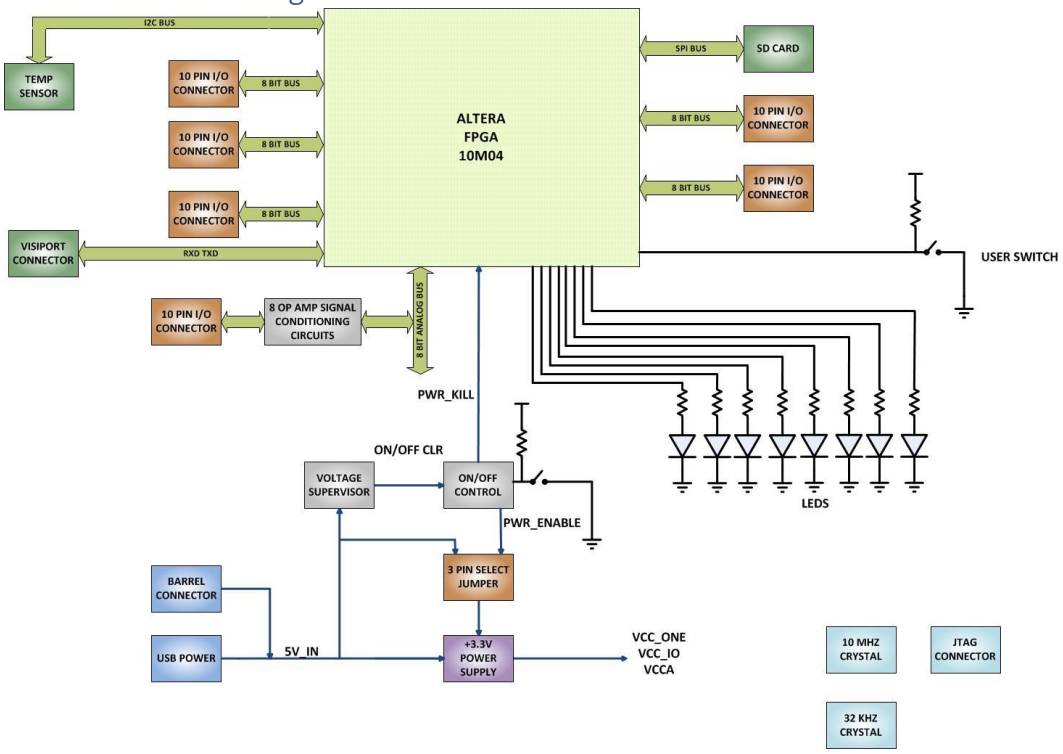
See section x for details on powering the board. Next, connect a JTAG Blaster or Altera USB Blaster to Jx for programming the MAX10 chip. Then, write some code, synthesize, and program the chip. See section x for details about programming the MaxProLogic board.

3 MaxProLogic Description

3.1 PCB Footprint

The PCB is 75mm x 50mm with four mounting holes (M2 metric screws) spaced as shown in the figure. These mounting holes are electrically isolated from all signals on the XEM6310. The two connectors (USB and DC power) overhang the PCB by approximately 1mm in order to accommodate mounting within an enclosure.

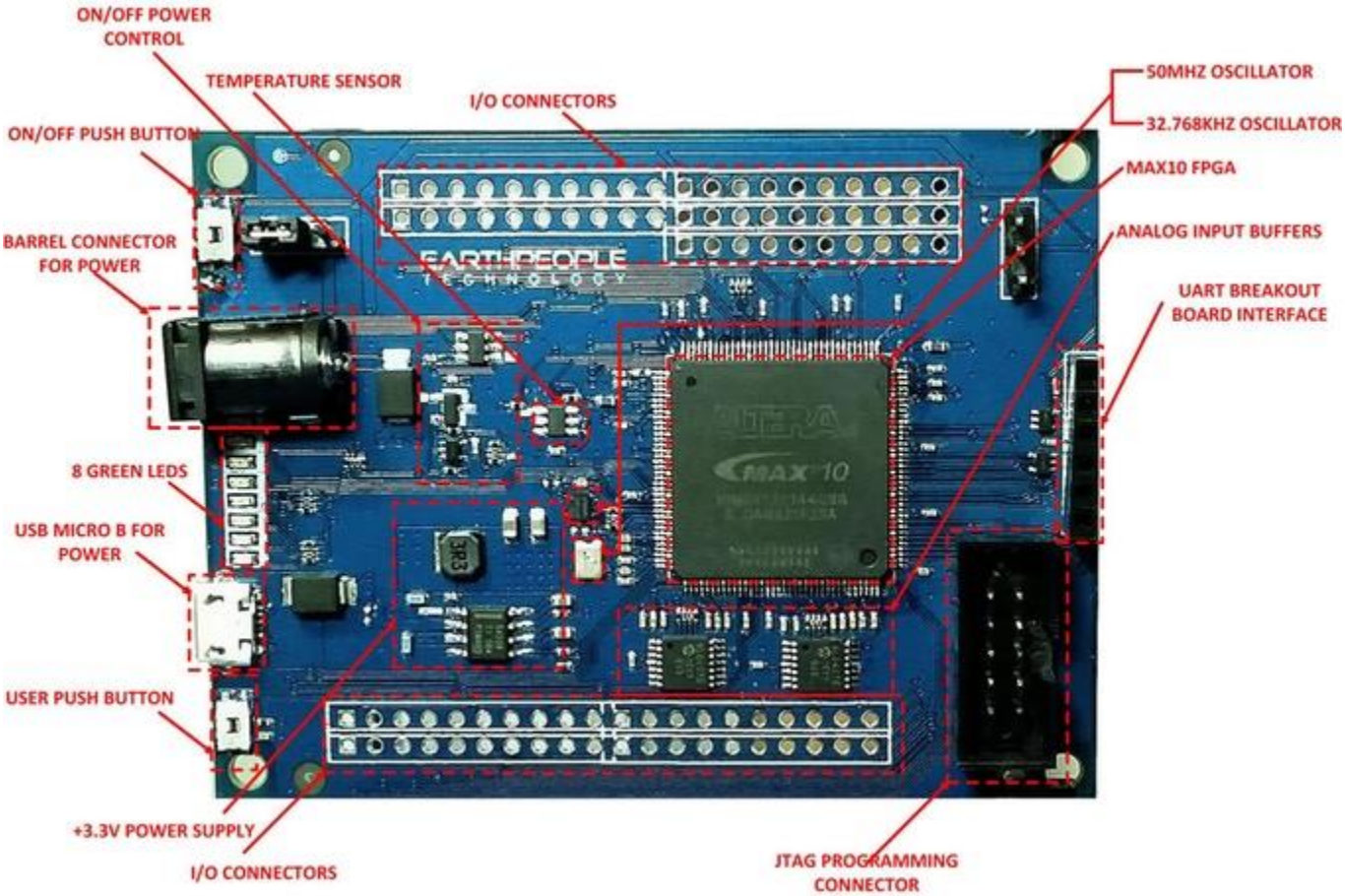
3.2 Functional Block Diagram



3.3 MAXPROLOGIC SPECIFICATIONS

- MAX 10 10M04SA FPGA FROM INTEL/ALTERA 4,000 Logic Elements;
- 2.2 Mbit On chip Flash; 189 Kbit On Chip SRAM 8 Analog Input
- Channels; 12 bit; 1MSamples/Second 65 Available I/O's at connectors
- 65 Inputs/Outputs available at connectors on board

- 8 Green User configurable LEDs 1 Power Pushbutton Switch; 1 User Configurable Pushbutton Switch
- On Board MicroSD Card Slot
- Two Power options: Standard USB (+5V @ 2Amp) Using Micro-B connector; 5mm Barrel Connector Accepts +12V @ 3Amp
- Switching Power Supply, Provides stable output under high load stress
- On/Off controller uses push button, optional bypass mode to allow board power up without On/Off control
- Two Clocks: 50MHz Oscillator; 32.768KHz Oscillator
- On board interface to Standard USB to Serial Adapters
- Temperature Sensor with +/-0.3C Resolution
- Standard Programming Connector fits any Altera USB Blaster



3.4 FPGA

The MaxProLogic includes the Intel/Altera 10M04SAE144C8G FPGA. It is an EQFP 144 pin package. This FPGA incorporates both the configuration flash and the ADC on the chip. This is different from conventional FPGAs as these two items are usually off chip. Access to the configuration flash is transparent to the user. The user does have access to the User Flash. This access is provided through the Altera MegaFunctions at the project level. Access to the ADC is also provided through the Altera MegaFunctions at the project level.

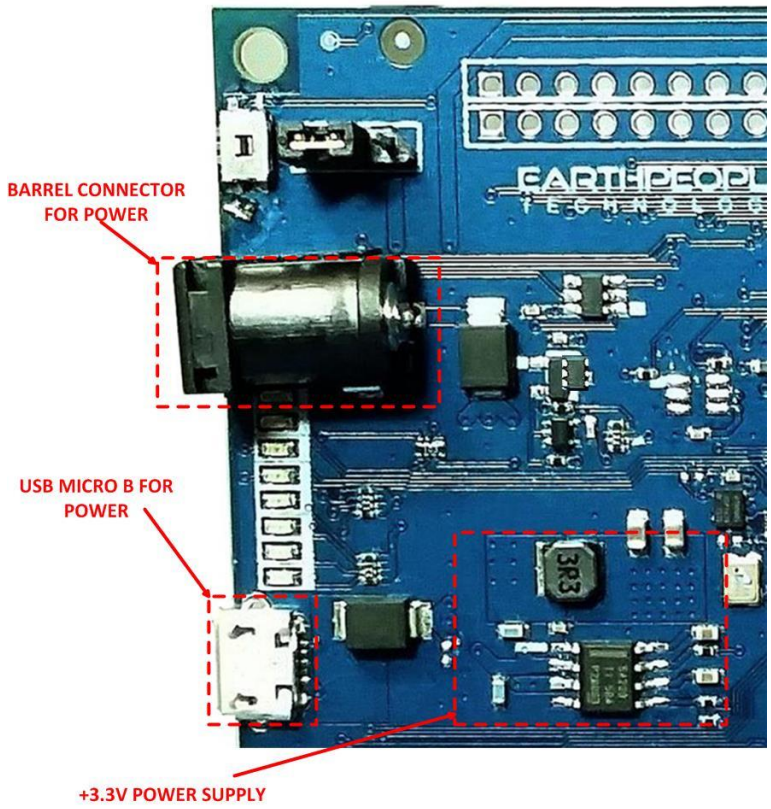
Parameter	**10M04SA**
LEs (Logic Elements)	4,000
Block memory (Kb)	189
User flash memory1 (KB)	16 – 156
18 x 18 multipliers	20
Phase-locked loops (PLLs)	2
Internal configuration	Dual
Analog-to-digital converter (ADC)	8 Channels, 1MSa/Sec
External memory interface (EMIF)	Yes
Inputs/Outputs	101

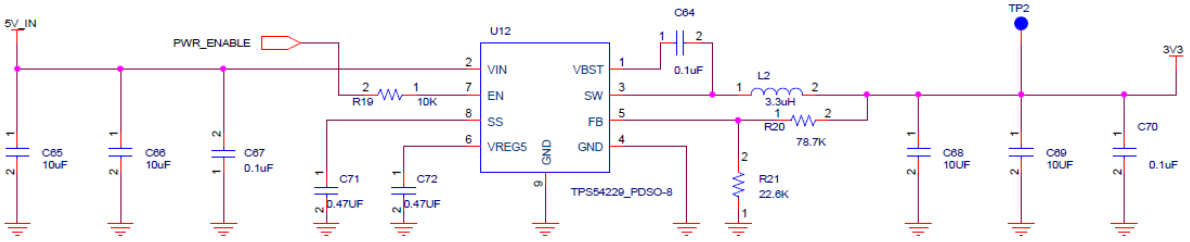
3.5 Power Supply

The MaxProLogic is designed to be operated from one of four different power sources:

- standard USB cable from Laptop/PC.
- +5 VDC wall charger (phone charger) through USB cable.
- +5.5 to +9 VDC supplied through the DC power jack.
- +5.5 to +9 VDC supplied through I/O connector pin.

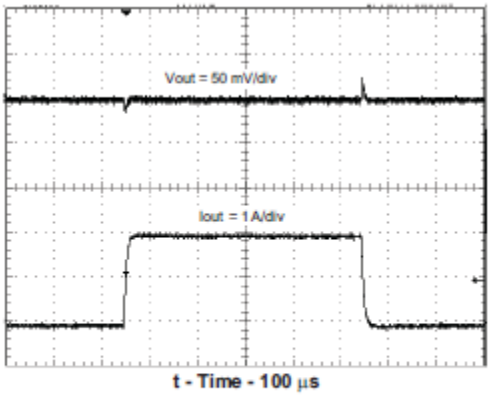
This provides power for a high-efficiency switching regulator on-board to provide +3.3 VDC. The power supply provides reliable power under dynamic loads and high frequency switching internal to the FPGA.





Buck Switching Regulator

The TPS54229 is a very stable synchronous buck converter. This allows it to provide a fast transient response. You can view this response here:

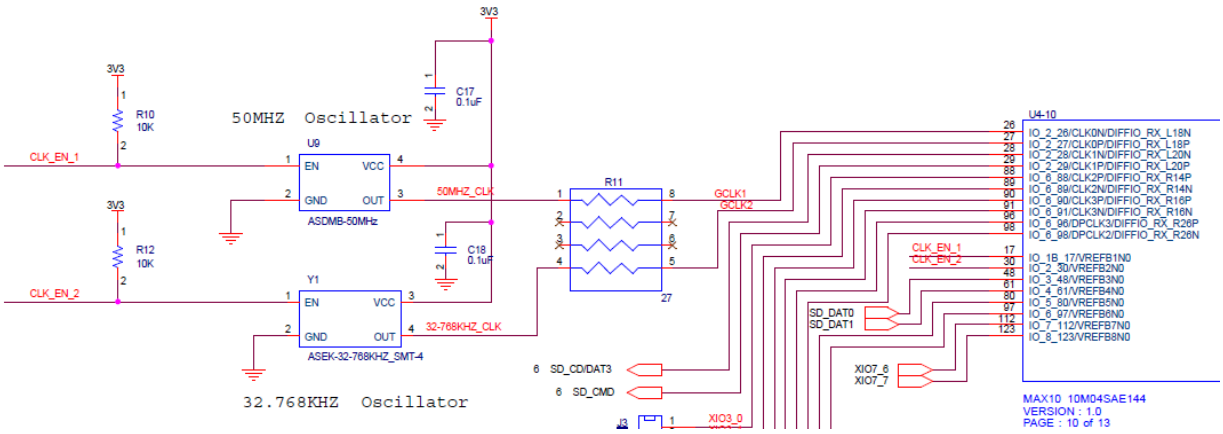


1.05-V, 50-mA to 2-A LOAD TRANSIENT RESPONSE

The TPS54229 has an Enable signal that allows the power supply to turn off. This signal is the net PWR_ENABLE. It is controlled by the On/Off controller MAX16054 or default. The operation of the On/Off controller is explained in section x. The default level for the PWR_ENABLE is high. The default level allows the power supply to turn on when power is applied to the board.

3.6 Clock Domains

There are two clock domains external to the MAX10 FPGA, 50 MHz and 32.768 KHz. The 50 MHz oscillator is a +3.3VDC device that provides a high speed clock to the FPGA. It is a CMOS device that provides a stable 50 MHz at ±50 ppm. This clock can be used directly in the user code or use it as an input to one of the PLL's internal to the FPGA. It is intended that this clock will drive the logic of the user code. If a different clock frequency is required in the user code, use the PLL scale up/down to produce the desired clocking.



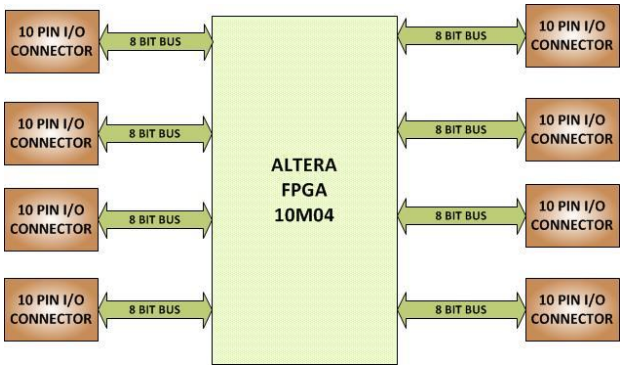
The 32.768 KHz oscillator is a +3.3VDC device that provides a stable source for low speed clocking. This oscillator can be used for lower speed functions such as PWM for motor control, Audio, or controlling LEDs.

Both oscillators have an enable signal that are individually controlled by the FPGA. The oscillators are operational when the enable signal is high. The oscillators are in a low power, no output mode when the enable signals are low.

Enable Signal	**High**	**Low**
CLK_EN_1 (50MHz Oscillator)	Oscillator output On	Oscillator output Off
CLK_EN_2 (32.768 KHz Oscillator)	Oscillator output On	Oscillator output Off

3.7 Digital I/Os

The MaxProLogic has eight 10 pin headers that provide 64 digital Inputs and Outputs. All of the I/O's are +3.3 VDC only. All I/O's connect directly from the FPGA to one of the ten pin headers.



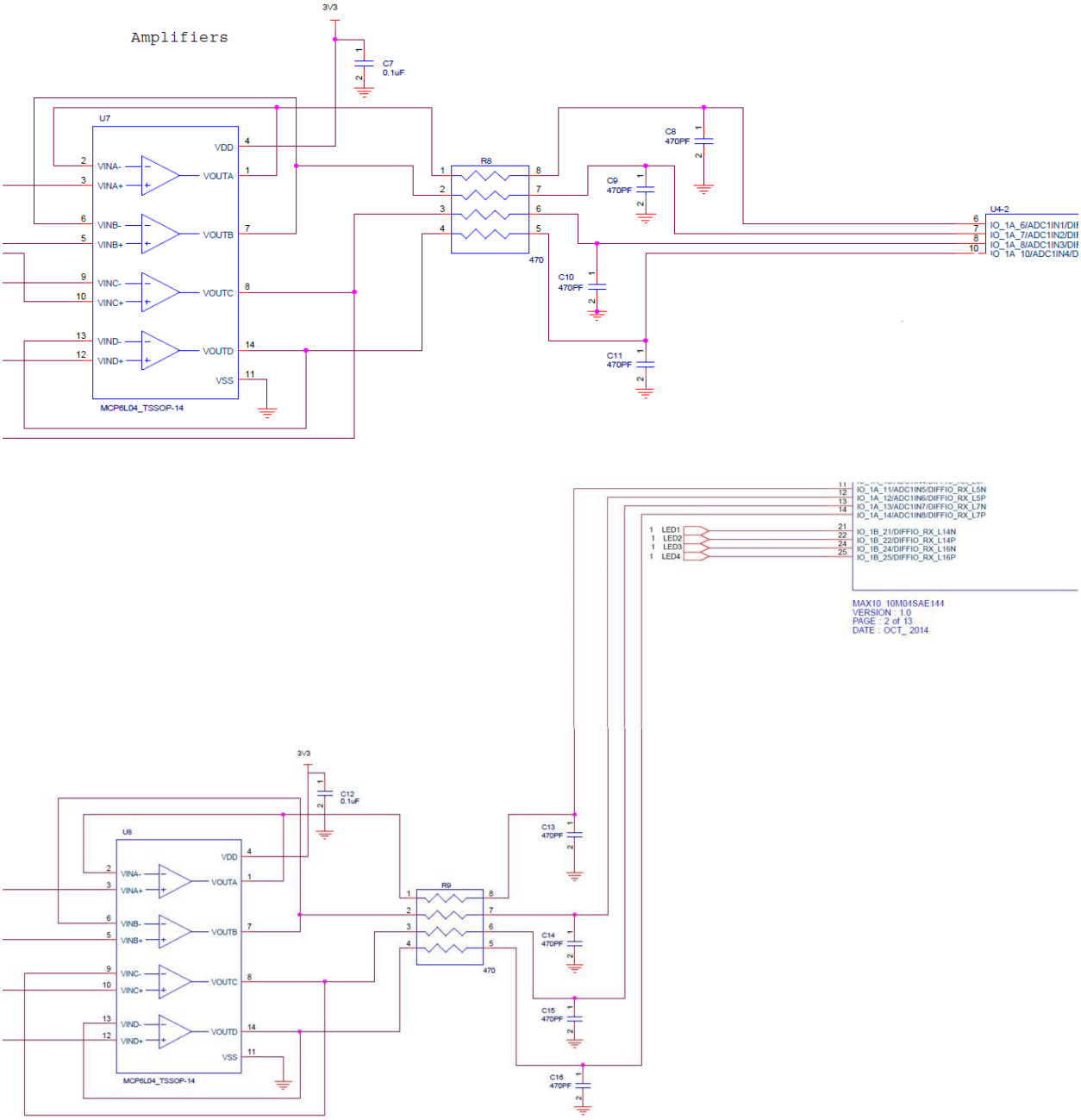
All I/O's are organized into separate banks of the FPGA. There are eight banks. These different banks provide different output speed technologies. Programmable Open Drain The optional open-drain output for each I/O pin is equivalent to an open collector output. If it is configured as

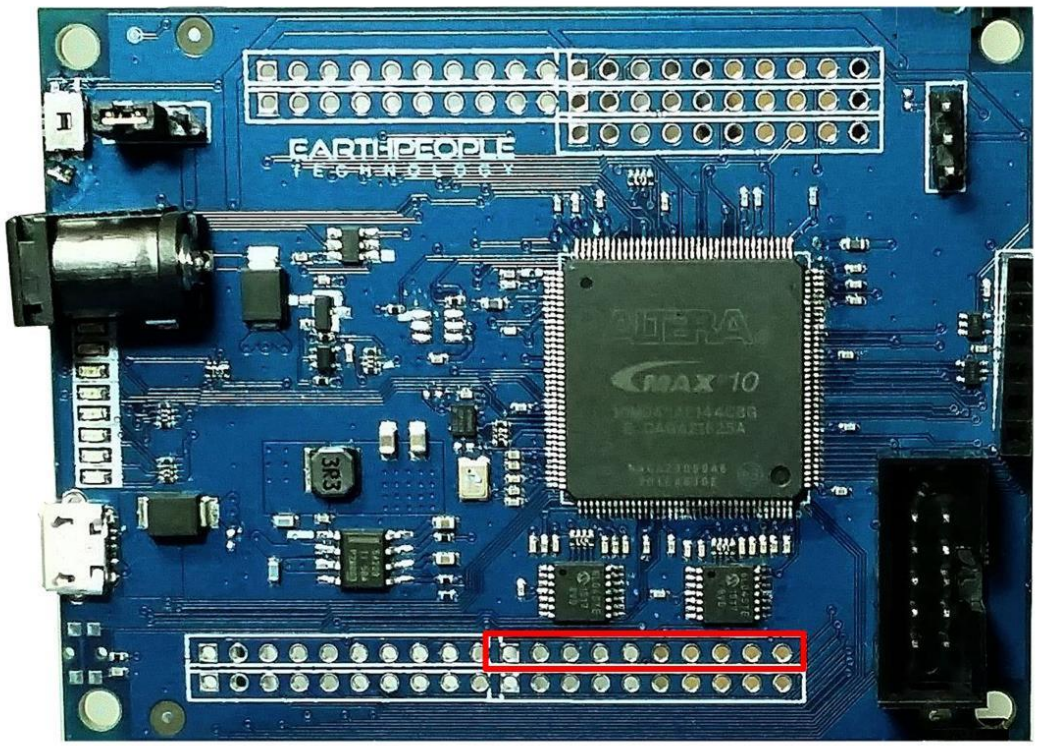
an open drain, the logic value of the output is either high-Z or logic low. Use an external resistor to pull the signal to a logic high. Programmable Bus Hold Each I/O pin provides an optional bus-hold feature that is active only after configuration. When the device enters user mode, the bus-hold circuit captures the value that is present on the pin by the end of the configuration. The bus-hold circuitry holds this pin state until the next input signal is present. Because of this, you do not require an external pull-up or pull-down resistor to hold a signal level when the bus is tri-stated. For each I/O pin, you can individually specify that the bus-hold circuitry pulls non-driven pins away from the input threshold voltage—where noise can cause unintended high-frequency switching. To prevent over-driving signals, the bus-hold circuitry drives the voltage level of the I/O pin lower than the VCCIO level. If you enable the bus-hold feature, you cannot use the programmable pull-up option. To configure the I/O pin for differential signals, disable the bus-hold feature. Programmable Pull-Up Resistor Each I/O pin provides an optional programmable pull-up resistor during user mode. The pull-up resistor weakly holds the I/O to the VCCIO level. If you enable the weak pull-up resistor, you cannot use the bus-hold feature. Programmable Current Strength You can use the programmable current strength to mitigate the effects of high signal attenuation that is caused by a long transmission line or a legacy backplane.

To provide maximum flexibility to system designers, all FPGA I/O are intrinsically bidirectional. Based on the I/O direction specified in the configuration file, each I/O can be configured as input-only, output-only, or bidirectional. While the output buffer of a bidirectional I/O has always included a dynamic output enable signal, MAX 10 FPGAs also include an input buffer enable/disable capability. When an input buffer is dynamically disabled, the buffer presents its last known state to the FPGA core fabric, so that no inputs inside the FPGA are left floating.

3.8 Analog Input

The MaxProLogic features an analog input of eight buffered channels. Each channel has its own 1MHz Unity Gain Amplifier to provide isolation and filtering. The Unity Gain Amplifier provides the best isolation between channels when using a Sample and Hold ADC and a high speed multiplexor.


Amplifiers
3V3
C7
0.1uF
U7
VDD
VINA-
VINA+
VOUTA
VINB-
VINB+
VOUTB
VINC-
VINC+
VOUTC
VIND-
VIND+
VOUTD
VSS
MCP6L04_TSSOP-14
R8
470
C8
470PF
C9
470PF
C10
470PF
C11
470PF
U4-2
IO_1A_6/ADC1IN1/DIF
IO_1A_7/ADC1IN2/DIF
IO_1A_8/ADC1IN3/DIF
IO_1A_10/ADC1IN4/D
IO_1A_11/ADC1IN5/DIFFIO_RX_L5N
IO_1A_12/ADC1IN6/DIFFIO_RX_L5P
IO_1A_13/ADC1IN7/DIFFIO_RX_L7N
IO_1A_14/ADC1IN8/DIFFIO_RX_L7P
LED1
LED2
LED3
LED4
IO_1B_21/DIFFIO_RX_L14N
IO_1B_22/DIFFIO_RX_L14P
IO_1B_24/DIFFIO_RX_L16N
IO_1B_25/DIFFIO_RX_L16P
MAX10_10M04SAE144
VERSION : 1.0
PAGE : 2 of 13
DATE : OCT_ 2014
3V3
C12
0.1uF
U8
R9
470
C13
470PF
C14
470PF
C15
470PF
C16
470PF

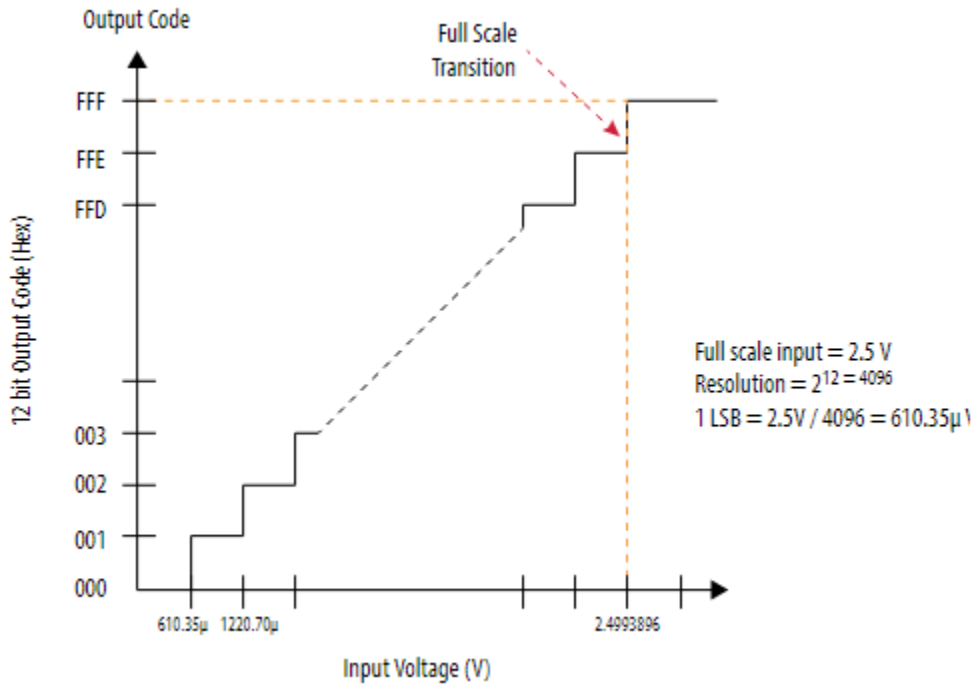


The ADC provide the MAX 10 devices with built-in capability for on-die temperature monitoring and external analog signal conversion. The ADC solution consists of hard IP blocks in the MAX 10 device periphery and soft logic through the Altera Modular ADC IP core. The ADC solution provides you with built-in capability to translate analog quantities to digital data for information processing, computing, data transmission, and control systems. The basic function is to provide a 12 bit digital representation of the analog signal being observed. The ADC monitors up to 8 single-ended external inputs with a cumulative sampling rate of one megasymbols per second (Msps).

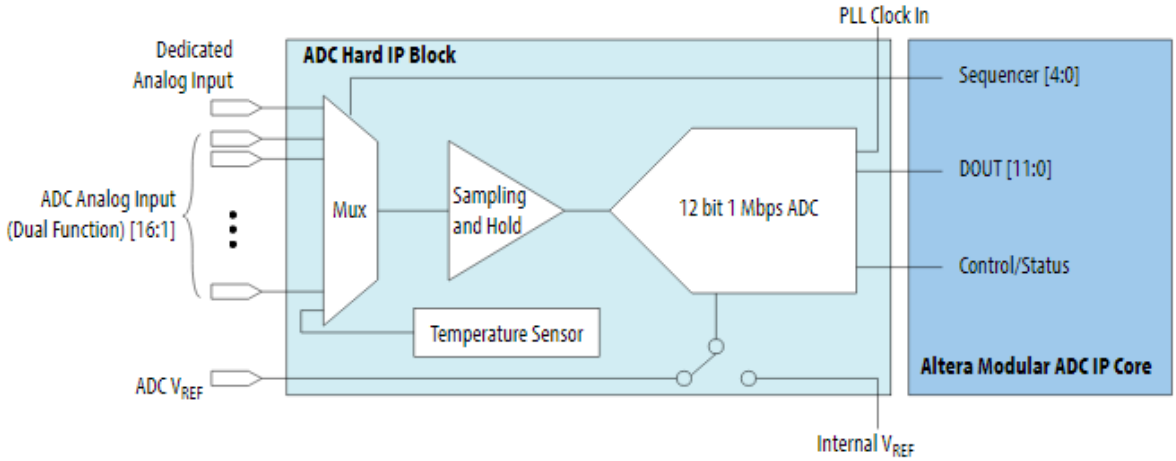
Intel MAX 10 ADC Conversion

The ADC in dual supply Intel® MAX® 10 devices can measure from 0 V to 2.5 V. In single supply Intel® MAX® 10 devices, it can measure up to 3.0 V or 3.3 V, depending on your power supply voltage.

In prescaler mode, the analog input can measure up to 3.0 V in dual supply Intel® MAX® 10 devices and up to 3.6 V in single supply Intel® MAX® 10 devices.
The analog input scale has full scale code from 000h to FFFh. However, the measurement can only display up to full scale – 1 LSB.
For the 12 bits corresponding value calculation, use unipolar straight binary coding scheme.



The Intel® MAX® 10 ADC is a 1 MHz successive approximation register (SAR) ADC. If you set up the PLL and Altera Modular ADC IP core correctly, the ADC operates at up to 1 MHz during normal sampling and 50 kHz during temperature sensing.



3.8.1 Voltage Representation Conversion

Use the following equations to convert the voltage between analog value and digital representation.

Conversion from Analog Value to Digital Code

$$\text{Digital Code}=(V_{IN}V_{REF})\times 212\text{Digital Code}=(VINVREF)\times 212$$

Conversion from Digital Code to Analog Value

$$\text{Analog Value}=\text{Digital Code}\times(V_{REF}212)\text{Analog Value}=\text{Digital Code}\times(VREF212)$$

3.8.2 Calculation Example for V_{REF} of **2.5 V**

Analog voltage value to digital code (in decimal), where signal in is 2 V:

$$\text{Digital Code}=(22.5)\times 4096=3277\text{Digital Code}=(22.5)\times 4096=3277$$

Digital code to analog voltage value, approximation to 4 decimal points:

$$\text{Analog Value} = 3277\times(2.54096)=2.0000$$

3.8.3 ADC Analog Input Pins

The analog input pins support single-ended and unipolar measurements.
The ADC block in Intel® MAX® 10 devices contains two types of ADC analog input pins:

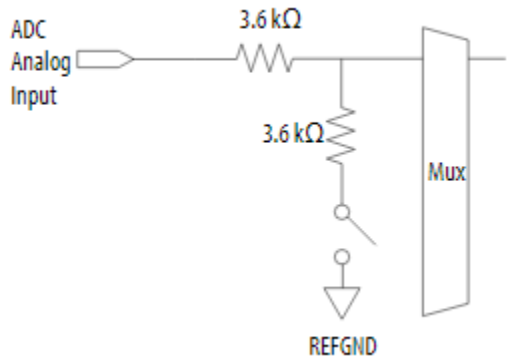
- Dedicated ADC analog input pin—pins with dedicated routing that ensures both dedicated analog input pins in a dual ADC device has the same trace length.
- Dual function ADC analog input pin—pins that share the pad with GPIO pins.

If you use bank 1A for ADC, you cannot use the bank for GPIO.

Each analog input pin in the ADC block is protected by electrostatic discharge (ESD) cell.

3.8.4 ADC Prescaler

The ADC block in Intel® MAX® 10 devices contains a prescaler function.
The prescaler function divides the analog input voltage by half. Using this function, you can measure analog input greater than 2.5 V. In prescaler mode, the analog input can handle up to 3 V input for the dual supply Intel® MAX® 10 devices and 3.6 V for the single supply Intel® MAX® 10 devices.



The prescaler feature is available on these channels in each ADC block:

- Single ADC device—channels 8 and 16 (if available)

3.8.5 ADC Clock Sources

The ADC block uses the device PLL as the clock source. The ADC clock path is a dedicated clock path. You cannot change this clock path.
For devices that support two PLLs, you can select which PLL to connect to the ADC. You can configure the ADC blocks with one of the following schemes:

- Both ADC blocks share the same clock source for synchronization.
- Both ADC blocks use different PLLs for redundancy.

If each ADC block in your design uses its own PLL, the Intel® Quartus® Prime Fitter automatically selects the clock source scheme based on the PLL clock input source:

- If each PLL that clocks its respective ADC block uses different PLL input clock source, the Intel® Quartus®Prime Fitter follows your design (two PLLs).

- If both PLLs that clock their respective ADC block uses the same PLL input clock source, the Intel®Quartus® Prime Fitter merges both PLLs as one.

In dual ADC mode, both ADC instance must share the same ADC clock setting.

Related information

PLL Locations, Intel® MAX® 10 Clocking and PLL User Guide

3.8.6 ADC Voltage Reference

Each ADC block in Intel® MAX® 10 devices can independently use an internal or external voltage reference. There is only one external VREF pin in each Intel® MAX® 10 device. Therefore, if you want to assign external voltage reference for both ADC blocks in dual ADC devices, share the same external voltage reference for both ADC blocks.
Intel recommends that you use a clean external voltage reference with a maximum resistance of 100 Ω for the ADC blocks. If the ADC block uses an internal voltage reference, the ADC block is tied to its analog voltage and the conversion result is ratiometric.

3.8.7 Creating Intel MAX 10 ADC Design

To create your ADC design, you must customize and generate the ALTPLL and Altera Modular ADC IP cores.
The ALTPLL IP core provides the clock for the Altera Modular ADC IP core.

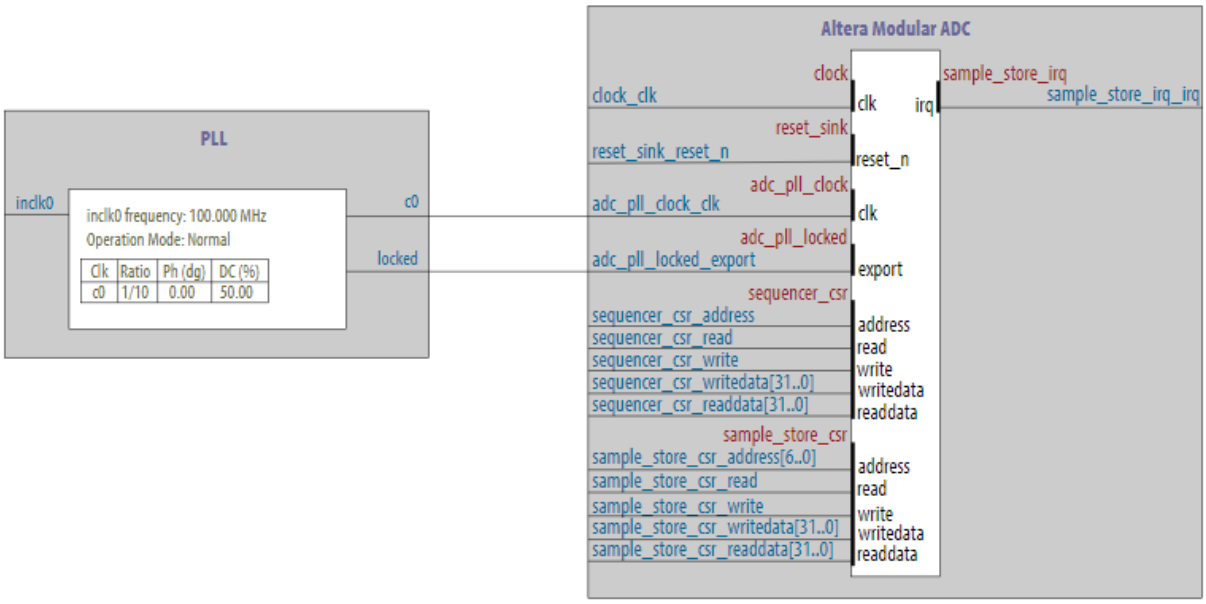
1. Customize and generate the ALTPLL IP core.
2. Customize and generate the Altera Modular ADC IP core.
3. Connect the ALTPLL IP core to the Altera Modular ADC IP core.
4. Create ADC Avalon slave interface to start the ADC.

3.8.8 Parameters Settings for Generating ALTPLL IP Core

Navigate through the ALTPLL IP core parameter editor and specify the settings required for your design. After you have specified all options as listed in the following table, you can generate the HDL files and the optional simulation files.
For more information about all ALTPLL parameters, refer to the related information.

3.8.9 Completing ADC Design

The ADC design requires that the ALTPLL IP core clocks the Altera Modular ADC IP core.
Generate the ALTPLL and Altera Modular ADC IP cores with the settings in the related information.



1. Create the design as shown in the preceding figure.
2. Connect the `c0` signal from the ALTPLL IP core to the `adc_pll_clock_clk` port of the Altera Modular ADCIP core.
3. Connect the `locked` signal from the ALTPLL IP core to the `adc_pll_locked_export` port of the Altera Modular ADC IP core.
4. Create the ADC Avalon slave interface to start the ADC.

Table 12. ALTPLL Parameters Settings.To generate the PLL for the ADC, use the following settings.

Tab	Parameter	Setting
Parameter Settings > General/Modes	What is the frequency of the inclk0 input?	Specify the input frequency to the PLL.

Table 12. ALTPLL Parameters Settings.To generate the PLL for the ADC, use the following settings.

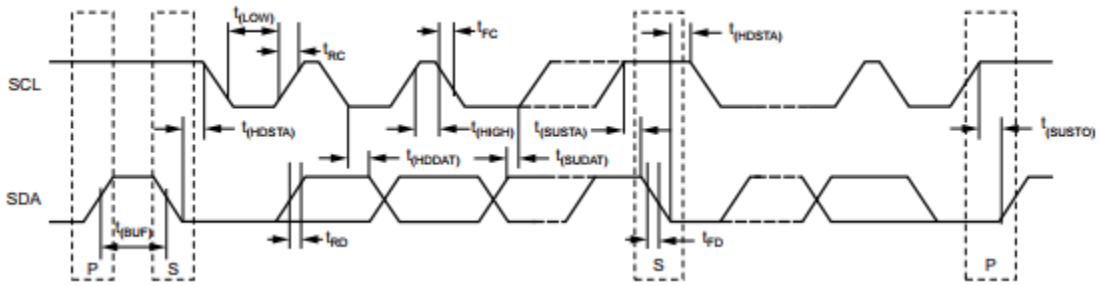
Tab	Parameter	Setting
Parameter Settings > Inputs/Lock	Create an 'areset' input to asynchronously reset the PLL	Turn off this option.
	Create 'locked' output	Turn on this option. You need to connect this signal to the `adc_pll_locked` port of the Altera Modular ADC or Altera Modular Dual ADC IP core.
Output Clocks > clk c0	Use this clock	Turn on this option.
	Enter output clock frequency	Specify an output frequency of 2, 10, 20, 40, or 80 MHz. You can specify any of these frequencies. The ADC block runs at 1 MHz internally but it contains a clock divider that can further divide the clock by a factor of 2, 10, 20, 40, and 80. Use this same frequency value in your Altera Modular ADC or Altera Modular Dual ADC IP core. You need to connect this signal to the `adc_pll_clock` port of the Altera Modular ADC or Altera Modular Dual ADC IP core. Different ADC sampling rates support different clock frequencies. For a valid sampling rate and clock frequency combination, refer to the related information.

3.9 Temperature Sensor

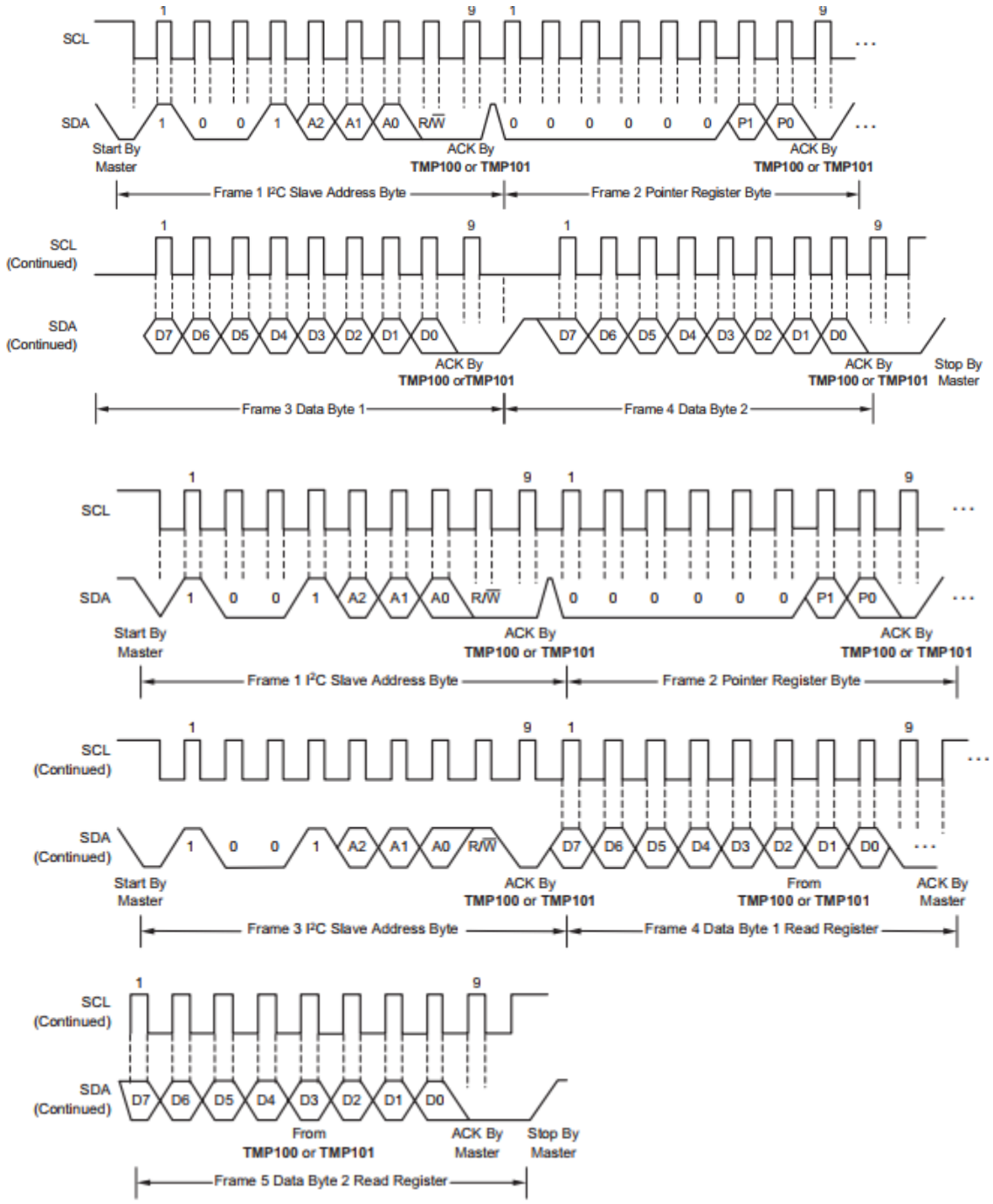
The MaxProLogic includes a Temperature Sensor from Texas Instruments, the TMP100. This digital sensor has an accuracy of ±1ºC. without requiring calibration or external component signal conditioning. Device temperature sensors are highly linear and do not require complex calculations or look-up tables to derive the temperature. The on-chip, 12-bit ADC offers resolutions down to 0.0625°C.

The sensor is accessed through an I2C bus. This bus is connected to the FPGA through pins 100 and 101. The bus uses 10K pullups on both the SDA and SCK signals.

Timing Diagrams The TMP100 and TMP101 devices are Two-Wire, SMBUs, and I 2C interface-compatible. Figure 6 to Figure 9 describe the various operations on the TMP100 and TMP101. The following list provides bus definitions. Parameters for Figure 6 are defined in the Timing Requirements section.

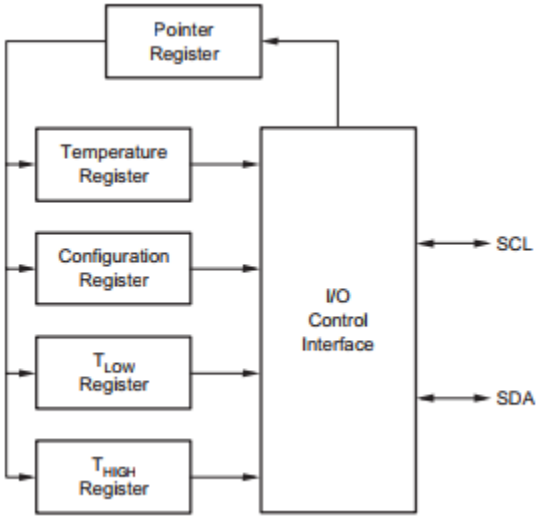


Bus Idle: Both SDA and SCL lines remain HIGH. Start Data Transfer: A change in the state of the SDA line, from HIGH to LOW, while the SCL line is HIGH, defines a START condition. Each data transfer is initiated with a START condition. Stop Data Transfer: A change in the state of the SDA line from LOW to HIGH while the SCL line is HIGH defines a STOP condition. Each data transfer is terminated with a repeated START or STOP condition. Data Transfer: The number of data bytes transferred between a START and a STOP condition is not limited and is determined by the master device. The receiver acknowledges the transfer of data. Acknowledge: Each receiving device, when addressed, is obliged to generate an Acknowledge bit. A device that acknowledges must pull down the SDA line during the Acknowledge clock pulse in such a way that the SDA line is stable LOW during the HIGH period of the Acknowledge clock pulse. Setup and hold times must be taken into account. On a master receive, the termination of the data transfer can be signaled by the master generating a Not-Acknowledge on the last byte that is transmitted by the slave. Figure 6. I 2C Timing Diagram



Programming 7.5.1 Pointer Register Figure 10 shows the internal register structure of the TMP100 and TMP101 devices. The 8-bit Pointer Register of the TMP100 and TMP101 devices

is used to address a given data register. The Pointer Register uses the two LSBs to identify which of the data registers respond to a read or write command. Table 4 identifies the bits of the Pointer Register byte. Table 5 describes the pointer address of the registers available in the TMP100 and TMP101 devices. The power-up reset value of P1 and P0 is 00.



Pointer Register Byte (pointer = N/A) [reset = 00h] Table 4. Pointer Register Byte

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	Register Bits	

Pointer Addresses of the TMP100 and TMP101 Registers Table 5. Pointer Addresses of the TMP100 and TMP101 Registers

P1	P0	TYPE	REGISTER
0	0	R only, default	Temperature Register
0	1	R/W	Configuration Register
1	0	R/W	T_{LOW} Register
1	1	R/W	T_{HIGH} Register

Temperature Register The Temperature Register of the TMP100 or TMP101 devices is a 12-bit, read-only register that stores the output of the most recent conversion. Two bytes must be read to obtain data, and are described in Table 6 and Table 7. The first 12 bits are used to indicate temperature, with all remaining bits equal to zero. Data format for temperature is summarized in Table 1. Following power-up or reset, the Temperature Register reads 0°C until the first conversion is complete. Table 6. Byte 1 of the Temperature Register

D7	D6	D5	D4	D3	D2	D1	D0
T11	T10	T9	T8	T7	T6	T5	T4

Table 7. Byte 2 of the Temperature Register

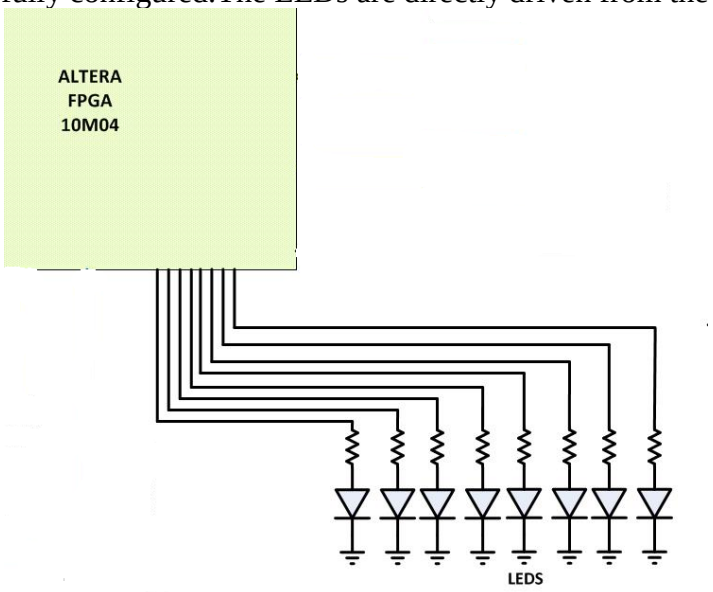
D7	D6	D5	D4	D3	D2	D1	D0
T3	T2	T1	T0	0	0	0	0

Configuration Register The Configuration Register is an 8-bit read and write register used to store bits that control the operational modes of the temperature sensor. Read and write operations are performed MSB-first. The format of the Configuration Register for the TMP100 and TMP101 devices is shown in Table 8, followed by a breakdown of the register bits. The power-up or reset value of the Configuration Register is all bits equal to 0. The OS/ALERT bit reads as 1 after power-up or reset value. Table 8. Configuration Register Format

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	OS/ALERT	R1	R0	F1	F0	POL	TM	SD

3.10 LEDs

The MaxProLogic includes eight user LEDs and One LED to indicate when the FPGA has been successfully configured.The LEDs are directly driven from the FPGA.



They use the +3.3V I/O's along with a 220 Ohm series resistor for each LED. This provides the following current through the LEDS

$$I_{LED} = \frac{V_O - V_F}{R}$$

$$I_{LED} = \frac{3.3V - 2.0V}{220}$$

$$I_{LED} = 5.9mA$$

The code to drive the LEDs is either zero (1'b0) or floating (1'bz). First, declare the LED as an output. In the example below, the vector LED is set to 'reg' because it is driven in an always block.

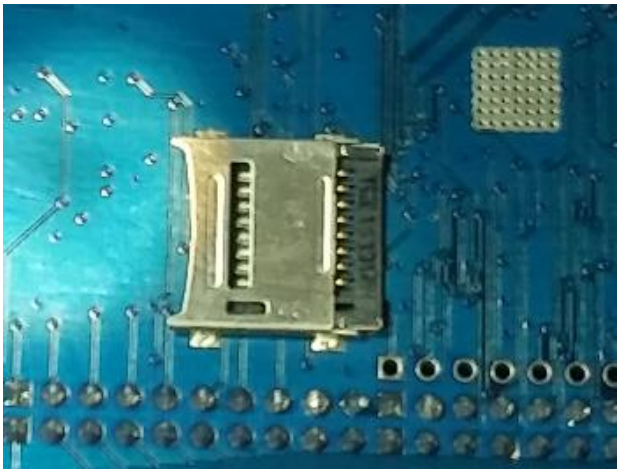
```
module EPT_10M04_AF_S2_Top (


    input  wire                 CLK_10MHZ,
    input  wire                 CLK_32KHZ,
    input  wire                 RST,

    output wire                 CLK_10MHZ_ENABLE,
    output wire                 CLK_32KHZ_ENABLE,

    output wire   [7:0]         XIO_1,        //
    input  wire   [7:0]         XIO_2,        //
    input  wire   [7:0]         XIO_3,        //
    output wire   [8:0]         XIO_4,        //

    input wire                  PWR_ENABLE,


    output reg     [7:0]          LED
    );
```

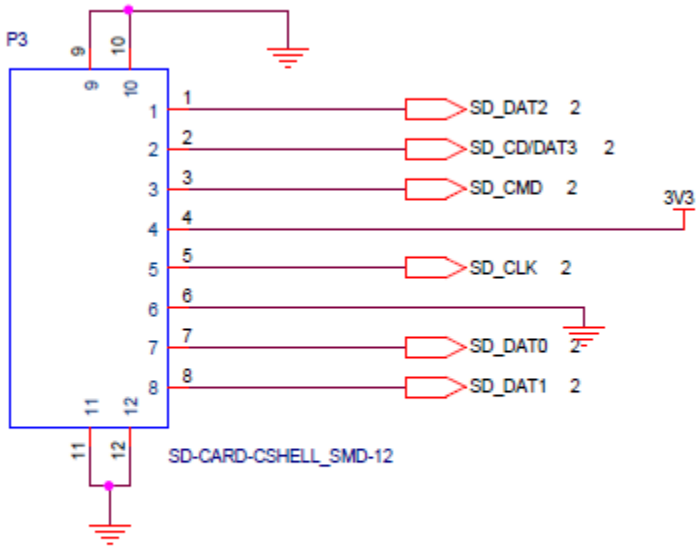
To turn the selected LED on, set the signal equal to 1'b0. This will apply a ground to the cathode side of the LED and allow current to flow through the circuit turning the LED on. To turn the selected LED off, set the signal equal to 1'bz. This will float the cathode side of the LED and no current will flow through the LED.

```
//------------------------------------------------
// Set the LED outputs
//------------------------------------------------
always @(posedge CLK_10MHZ or negedge RST)
begin
  if(!RST)
      LED <= 8'hz;
  else
  begin
    if(state[LOAD_LEDS])
    begin
      if ( led_reg[0] )
          LED[0] = 1'b0;
      else
          LED[0] = 1'bz;

      if ( led_reg[1] )
          LED[1] = 1'b0;
      else
          LED[1] = 1'bz;

      if ( led_reg[2] )
          LED[2] = 1'b0;
      else
          LED[2] = 1'bz;
```

3.11 SD Card Interface

The SD Connector uses a standard cell phone SD Card Connector. This connector has a mechanism that opens out away from the board on a hinge and allows the card to be inserted into the hinged door. Close the hinged door and pull it towards the front of the MaxProLogic to secure it.





SD Card Connector

3.11.1 SD Card Protocol

There's a ton of information out there on using the MMC/SD SPI protocol to access SD cards but not much on the native protocol. This page hopes to rectify that with information helpful to those implementing a SD host or trying to understand what they're seeing on an oscilloscope.

3.11.2 References

- The full MMC Specification is available as JESD84-A44 from the JEDEC website
- The SD Simplified Specification from the SD Card Association covers most of the protocol
- The full SD specification is available after joining the SD Card Association

3.11.3 Compatibility

MMC, SD, and SDHC cards are broadly compatible at the electrical and framing level. A properly designed controller should be able to handle them all. Some differences are:

- MMC cards are available in both High Voltage (2.7 - 3.6 V) and Dual Voltage (2.7 - 3.6 and 1.70 - 1.95 V)
- MMC is designed to support multiple cards on the same bus

- During initialisation MMC cards are clocked at 400 kHz or less
- MMC, SD, and SDHC all have different initialisation sequences

3.11.4 Protocol

The protocol is a strict master/slave arrangement where data is clocked synchronously from the host to the card or from the card to the host over digital lines. Commands are sent from the host to the card and all commands have either no response, a 48 bit response, or a 136 bit response. Some commands may also start a data transfer to or from the card.

There are three types of signal:

- CLK, carrying the clock signal from the host
- CMD, carrying commands from the host and responses from the card
- DAT, carrying data from the host or data from the card

There may be 1, 4, or 8 DAT lines. SD Cards can run at 0 - 25 MHz in Default Mode or 0 to 50 MHz in High-Speed Mode. MMC cards come in different grades that can run at up to 20, 26, or 52 MHz. No matter what all cards start up in 3.3 V, single DAT, and low speed mode with any other features negotiated during the initialisation.

3.11.5 Physical layer

All communication are at 3.3 V logic levels with 3.3 V being a high and 0 V being a low. CLK comes from the host and idles low. CMD and DAT are bidirectional and idle high. All are driven in a push/pull mode for speed.

Data is clocked into the host or card on the rising edge of CLK and changes on the falling edge. This is equivalent to the SPI (0, 0) mode.

3.11.6 Framing

The framing is a bit unusual. It feels like it was written by a embedded software engineer instead of a hardware or protocol engineer as the framing and use of CRCs is unusual and inconsistent. The advantage is that the framing maps through to a software only implementation pretty well.

All transfers start with a zero start bit and finish with a one stop bit. A card may signal that it is still working on the response by keeping the CMD line high until the response is ready.

All commands are 48 bits (6 bytes) long and all responses are either 48 bits (6 bytes) or 136 bits (17 bytes) long. The 48 bit transfers can be thought of as an 8 bit message ID, 32 bit argument, and 8 bit checksum.

Bytes are transferred most significant bit first. Words are transferred most significant byte first.

3.11.7 Commands and Responses

A command or response has the following format:

Bit	#	Value	Name
47	1	0	Start bit
46	1	1 for commands, 0 for responses	Transmitter bit
45-40	6		Command ID
39-8	32		Argument
7-1	7		CRC
0	1	1	Stop bit

The CRC is a 7 bit CRC with polynomial $x^7 + x^3 + 1$. A table driven form can be found in the Linux kernel under lib/crc7.c. Bitwise forms may be generated using pycrc with the parameters

```
--width=7 --poly=9 --reflect-in=0 --reflect-out=0 --xor-out=0 --xor-in=0 --generate c
--algorithm=bit-by-bit-fast
```

such as

```
for (int b = 0; b < 8; b++)
{
    uint bit = crc & 0x40;

    if ((data & 0x80UL) != 0)
    {
        bit ^= 0x40;
    }

    data <<= 1;
    crc <<= 1;

    if (bit != 0)
    {
        crc ^= 0x09;
    }
}
```

Note that the final CRC must be ANDed with 0x7F.

The CRC seed is zero and is calculated over the start, transmitter, command ID, and argument fields. The resulting CRC is compared for equality with the CRC from the message.

Some examples are:

An APP_CMD (55) command that prefixes a SD specific command
Bytes: `0x770000000065`
Bits: `0 1 110111 00000000000000000000000000000000 0110010 1`
Fields:

- Start bit = 0
- Transmitter = 1
- Command = 55 (decimal)
- Argument = 00000000
- CRC = 0x32
- Stop bit = 1

The CRC can be generated by feeding 0x77, 0x00, 0x00, 0x00, 0x00 into the CRC function above.

The response to the APP_CMD
Bytes: `0x370000012083`
Bits: `0 0 110111 00000000000000000000000100100000 1000001 1`
Fields:

- Start bit = 0
- Transmitter = 0 (this is a response)
- Response = 55 (decimal)
- Argument = 00000120
- CRC = 0x41
- Stop bit = 1

The response to CMD3 SEND_RELATIVE_ADDR
Bytes: `0x03B368050019`
Bits: `0 0 000011 10110011011010000000010100000000 0001100 1`
Fields:

- Start bit = 0

- Transmitter = 0 (this is a response)
- Response = 3 (decimal)
- Argument = 0xB3680500
- CRC = 0x0C
- Stop bit = 1

Note that this cards Relative Card Address (RCA) is 0xB368

3.11.8 Data

Data has the following format:

Bit	#	Value	Name
4113	1	0	Start bit
4112-17	512*8		Data bits
16-1	16		CRC
0	1	1	Stop bit

Note that this is for a typical transfer of a block of 512 bytes. The host knows from the command that was sent how many bytes to expect back. There is no other way of knowing the message length.

The CRC is is the ITU-T V.41 16 bit CRC with polynomial 0x1021. A table driven version can be found in the Linux kernel under lib/crc-itu-t.c. Note that there is no such thing as 'the' CRC16 so make sure you get the right one.
Unlike the commands or responses the CRC is calculated over all of the data bytes and does not include the start bit. The calculated CRC is checked for equality with the received CRC.

PENDING: Add an example data message with CRC.

3.11.9 Handover

The CMD and DAT lines are bidirectional. Handover occurs at the end of a command where both the host and the card switch to input mode for two clocks before the card starts driving in push/pull mode. Some commands must be responded to in a fixed number of clocks but most allow an arbitrary time before the response must start.

3.11.10 Initialisation

To initialise a SD or SDHC card, send the following:

- Write 74 clocks with CMD and DAT high

- Write `CMD0 GO_IDLE_STATE`. This will reset the card.
- Write `CMD8 SEND_IF_COND` for 3.3 V parts. If any SDHC cards are

present then you will get a wired-OR response with 0x3F as the command and 0xFF as the CRC and stop bit. Note that this must be sent or SDHC cards will not respond to the following steps

- Write `CMD55 APP_CMD`
- Receive a 55 response
- Write `ACMD41 SD_SEND_OP_COND`
- Expect a wired-OR response with 0x3F as the command and 0xFF as the CRC and stop bit
- Check the ready bit in the previous response. If the card is not

ready then repeat the CMD55/ACMD41 until it is

- Write `CMD2 ALL_SEND_CID`
- Expect a wired-OR response with 0x3F as the command and 0xFF as the CRC and stop bit
- Write `CMD3 SEND_RELATIVE_ADDR`
- Expect a 3 response. The upper two bytes of the argument is the

Relative Card Address (RCA) which is used in the next step

- Write `CMD7 SELECT_CARD` with the RCA
- Expect a 7 response

The card is now selected and ready to transfer data. See Figure 4-1 'SD Memory Card State Diagram' in the simplified spec for more information.

See section 4.7.4 'Detailed Command Description' in the simplified spec for more information on the commands and responses.

MMC cards are initialised using a similar but different method.

An example flow captured from Linux on a SC2440 is:

Phase	Command	Response	Notes
CMD0	40000000000 95	None	
CMD55	77000000000 65	370000012083	

ACMD41 SEND_OP_COND	690010000005F	3F00FF8000FF	Card is busy
CMD55	770000000065	370000012083	
ACMD41	690010000005F	3F00FF8000FF	Card is still busy
CMD55	770000000065	370000012083	
ACMD41 SEND_OP_COND	690010000005F	3F80FF8000FF	Card is ready
CMD2 ALL_SEND_CID	420000000004D	3F1D4144534420202010A0400BC10088ADFF	
CMD3 SEND_RELATIVE_ADDR	430001000007F	03B368050019	RCA of 0xB368

Note the missing CMD8 as this controller does not support SDHC. I didn't capture the final CMD7.

3.11.11 Reading

Once initialised reading from a card is straight forward.

To read a single page:

- Send `CMD17 READ_SINGLE_BLOCK` with the offset to read from as the argument
- Receive on the DAT lines
- Send `CMD12 STOP_TRANSMISSION` once the block has been received

To read consecutive pages:

- Send `CMD18 READ_MULTIPLE_BLOCK` with the starting offset as the argument
- Receive as many blocks as you want on the DAT lines
- Send `CMD12 STOP_TRANSMISSION` once the done

Note that there will be a response to these commands and the response may be interleaved with the data. See Figure 3-3: (Multiple) Block Read Operation for more information.

On SDHC cards the offset is in terms of 512 byte blocks. On SD cards the offset is in bytes and the number of bytes received depends on`CMD16 SET_BLOCKLEN`

3.12 Power Input

The MaxProLogic is designed to be operated from one of four different power sources:

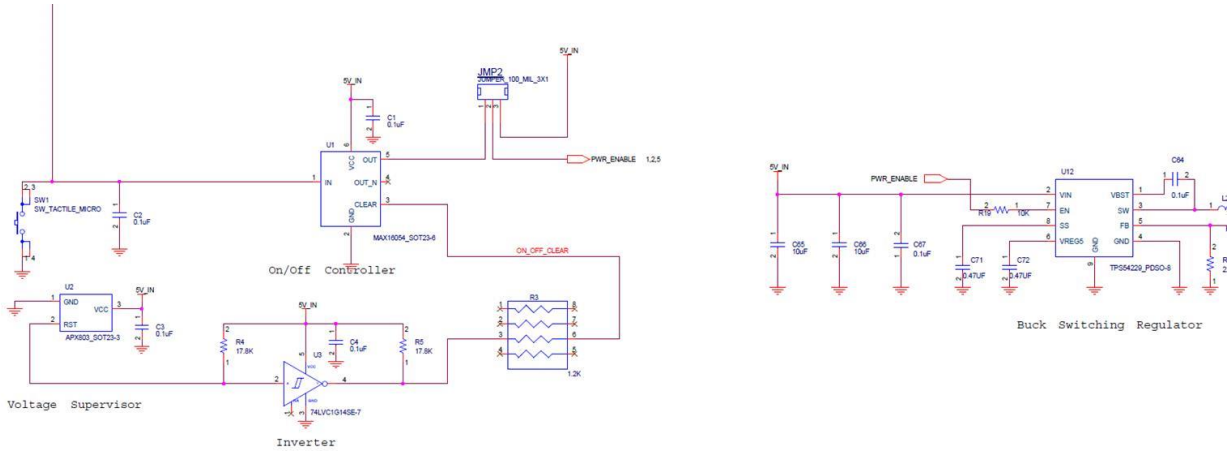
- Standard USB cable from Laptop/PC.
- +5 VDC wall charger (phone charger) through USB cable.
- +5.5 to +9 VDC supplied through the DC power jack.
- +5.5 to +9 VDC supplied through I/O connector pin.

This provides power for a high-efficiency switching regulator on-board to provide +3.3 VDC. The power supply provides reliable power under dynamic loads and high frequency switching internal to the FPGA.

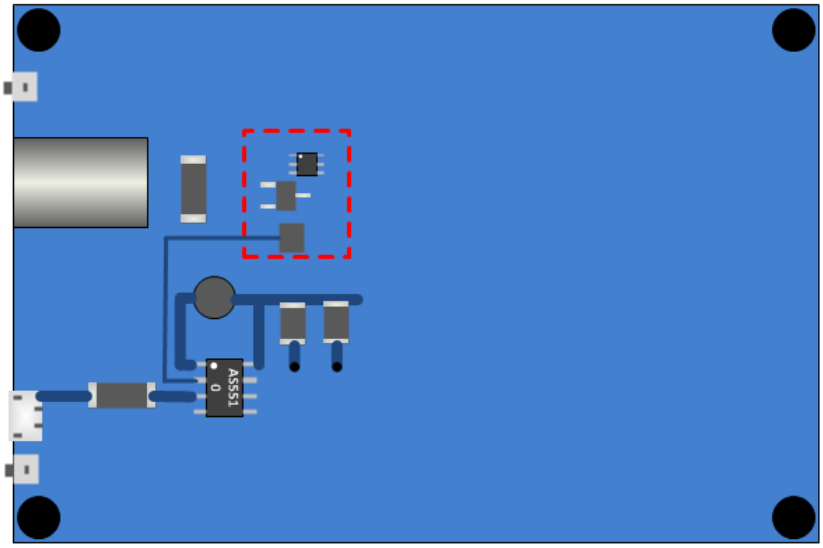
3.13 On/Off Control

The MaxProLogic is equipped with an On/Off circuit. This circuit consists of a push button switch that is momentary contact and a MAX16054 controller chip. The controller senses the change in state of the momentary switch and drives the output “PWR_ENABLE” signal to the +3.3 VDC power supply. The enable pin of the TPS54229 has the following truth table:

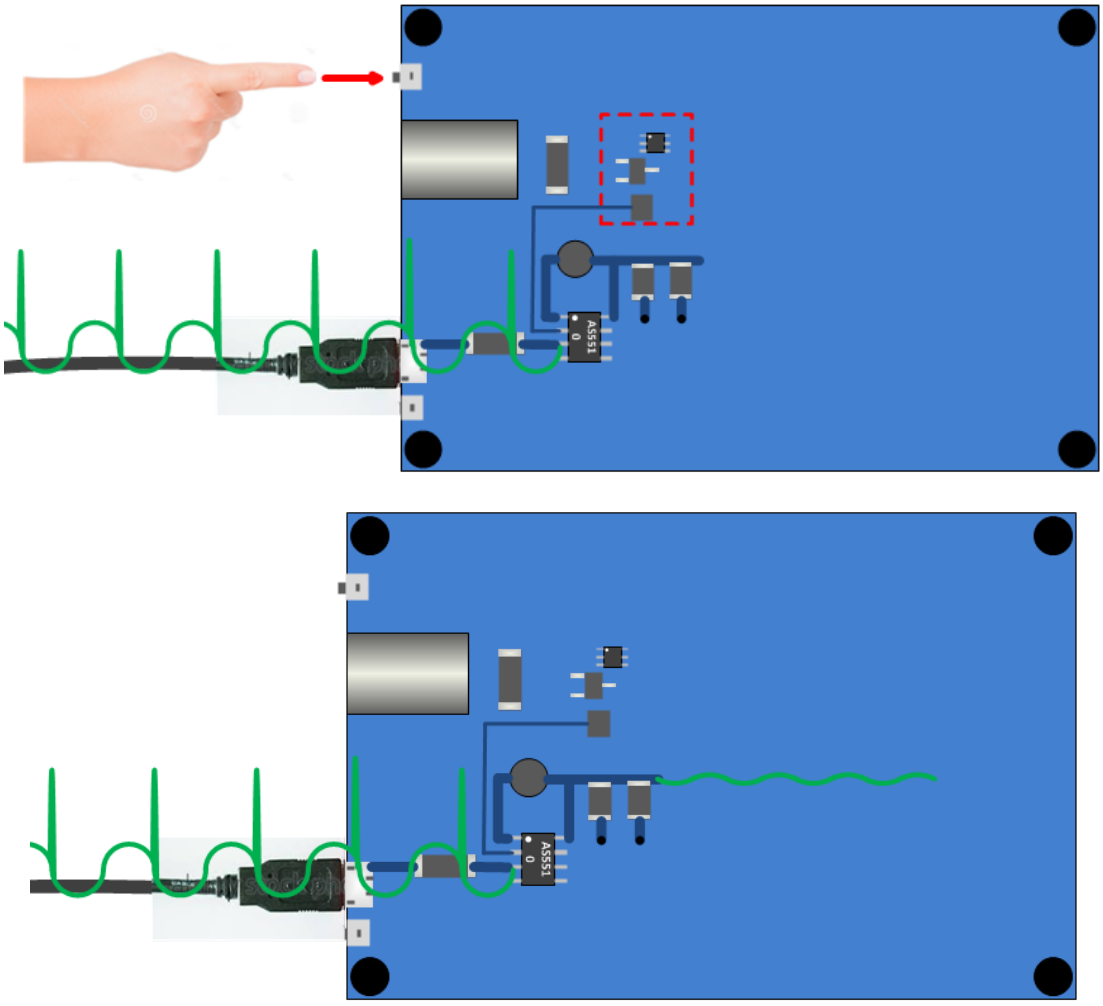
Enable Pin	**Output of TPS54229**
High	+3.3 V
Low	0V



Use the Pushbutton SW1 to turn the power on to the MaxProLogic.



The TPS54229 Synchronous Buck Regulator will take any noisy input power and provide a smooth stable output. It has a fast transient response with a large inductor on the output.



The MAX16054 Controller chip also has a ‘CLEAR’ input. This input allows secondary device to cause a shutdown. The CLEAR input has the following truth table:

Clear Pin	**Output of TPS54229**
High	+3.3 V
Low	0V

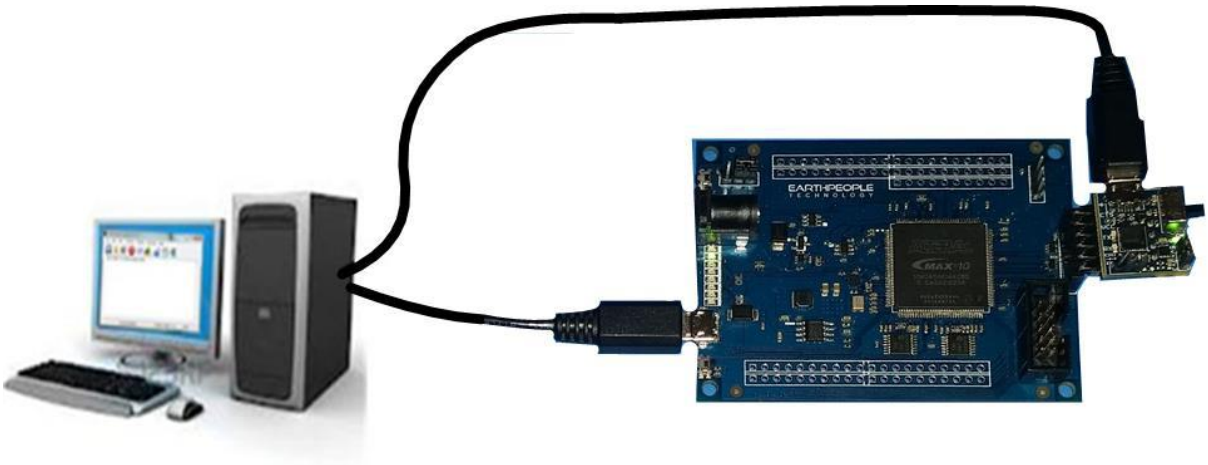
The On/Off controller can be bypassed and allow the MaxProLogic to power up whenever power is applied. The JMP2 provides the bypass. Set the jumper to the ‘1’ position to use the On/Off

pushbutton switch. Uses the ‘2’ position to allow the MaxProLogic to power up with board power.

3.14 Communications Interface

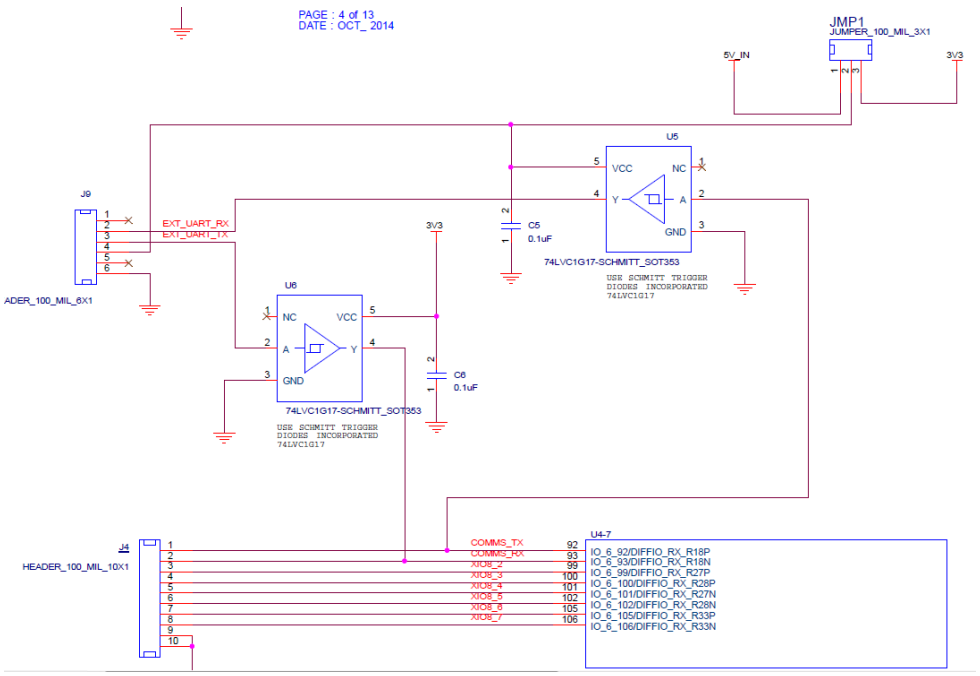
The MaxProLogic is equipped with a communications port that is compatible with FTDI Breakout Boards. It is a 6 pin female header that connects directly with most Breakout boards. The pinout:

Communications Pin #	Signal
1	NC
2	RX
3	TX
4	VCC
5	NX
6	GND



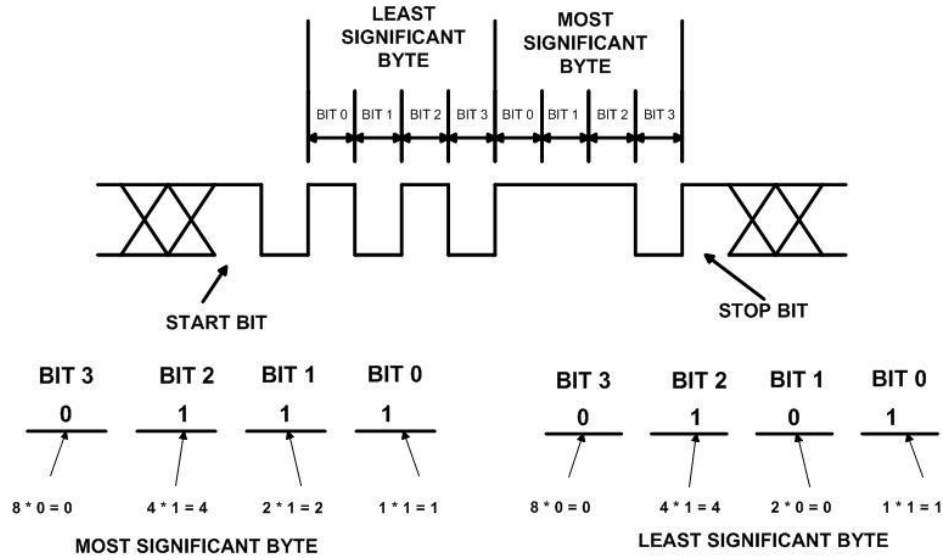
The Communications Connector provides a path between the FTDI Breakout Board and the MAX10 FPGA. It brings both the RX and TX signals into the FPGA at the following pins.

FPGA Pin #	Signal
92	RX
93	TX



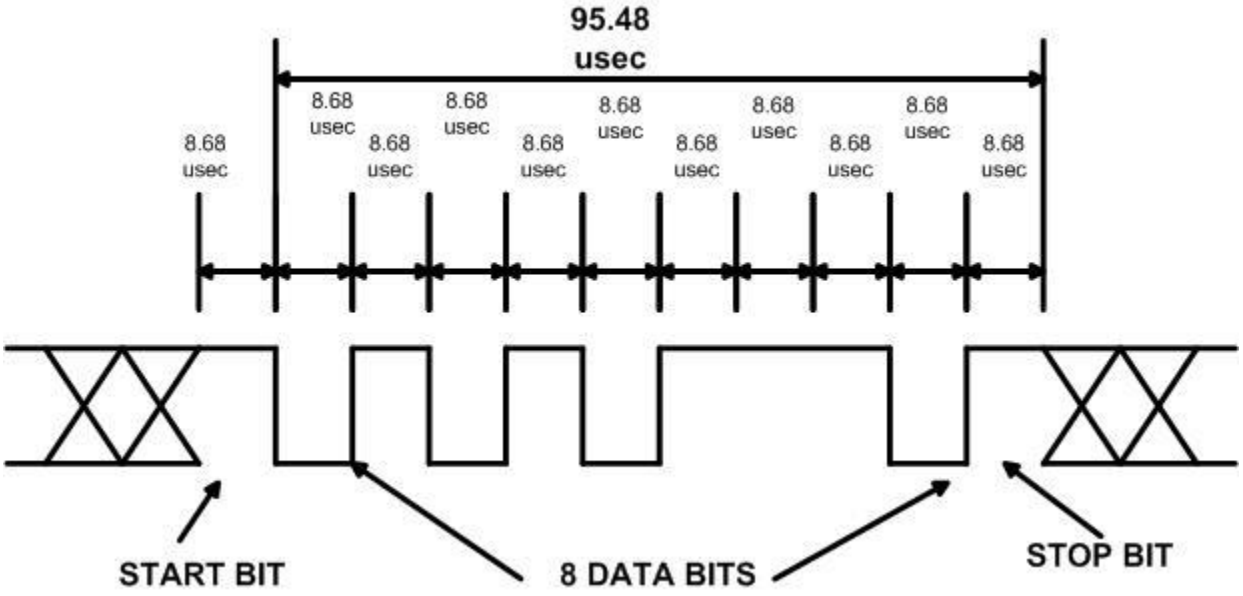
The MaxProLogic Communications Interface is compatible with both +5V and +3.3V Breakout Boards. The JMP1 provides a user selection jumper to select the +5V and +3.3V interface. There are two 74LVC1G17 Schmitt Trigger chips to provide voltage level compatibility with the FPGA.

The FPGA must run Verilog code to perform the full duplex UART communications. The code must decode in the incoming RX signal and produce the outgoing TX signals with an accuracy of greater than 99%. This means up to 1% error is acceptable. The UART signals are based on a protocol. Where the UART protocol indicates one start bit, eight data bits and one stop bit.

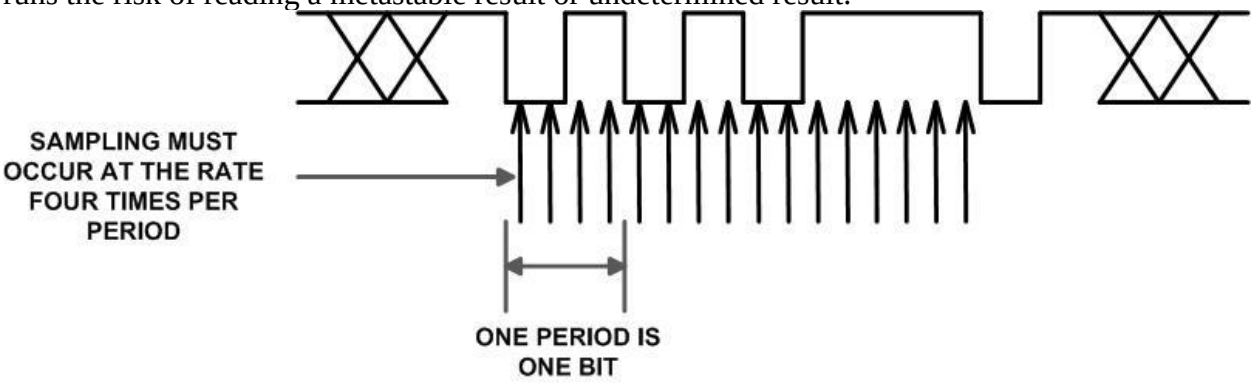


Each bit of the protocol must assert in the allocated time slot for each bit. What this means is that the clock used to time each bit of the protocol is embedded in the data. Both the sending device and receiving device must use the same clock rate. This embedding of the clock into data rate is called the baud rate.

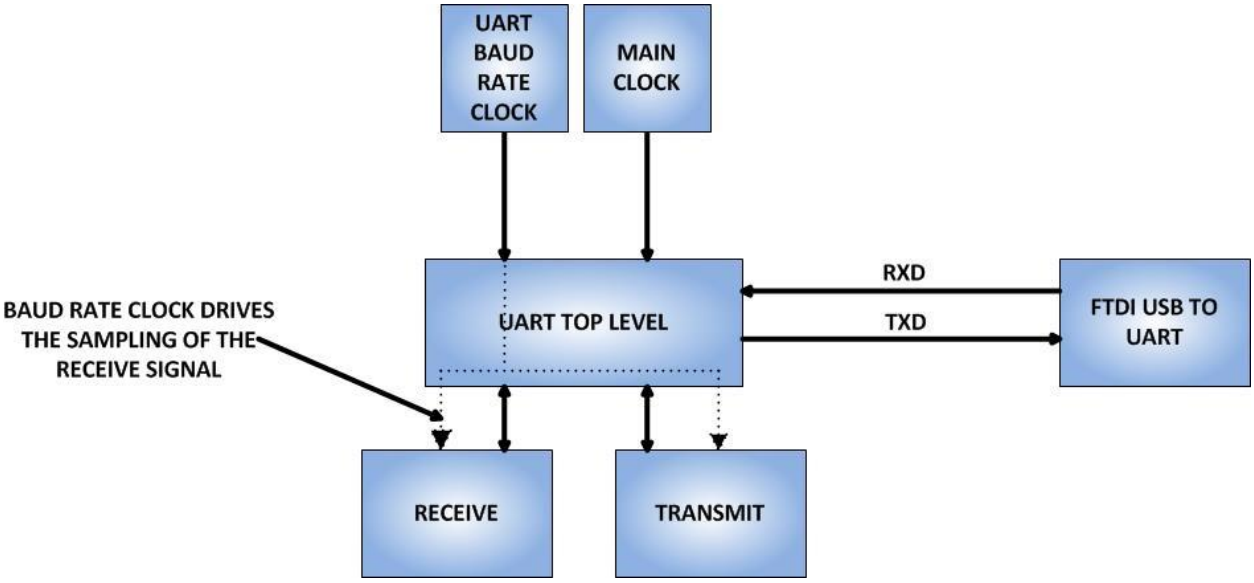
The baud rate describes the data rate in bits per second. The timing diagram for a baud rate of 115,200 is 8.68 microseconds per bit.



For the incoming RX signal, the FPGA must sample each bit of the protocol at four times per bit. This sampling rate is critical as the sending device clock and the receiving device clock are not synchronized. If the sampling rate is too low, a transition may be missed. If the sampling rate is only two times greater than the period of the receive device clock, samples could fall in the rising of falling edge of the received signal. Any time a sample occurs on the rising or falling edge, it runs the risk of reading a metastable result or undetermined result.



The Verilog code is written so that a special Baud Rate Clock is used to drive the UART Receive Block.

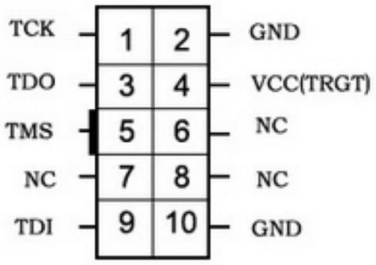


This code will use the Baud Rate Clock to drive a conditional if statement. The if statement will branch when a transition occurs and records the state of the signal (1 or 0). The Baud Rate Clock must be set to four times the incoming signal bit rate.

3.15 JTAG Interface

The MaxProLogic has a 5x2 header for use in programming the MAX10 FPGA via JTAG. The connector is located in the bottom right corner of the MaxProLogic. It is shrouded and keyed to allow easier insertion.

This connector uses the standard Altera Blaster connector pinout.

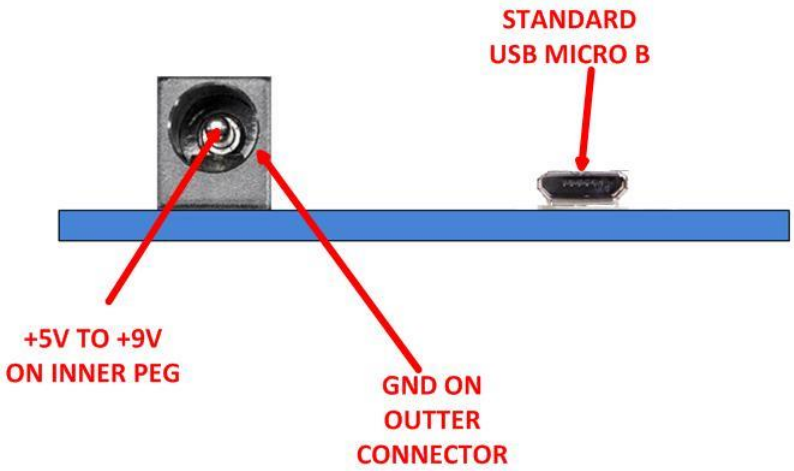


The VCC(TRGT) is set to +3.3V on the MaxProLogic. There are no jumper settings to make in order to program the MAX10 FPGA. Just connect a compatible Blaster to the connector and the PC, then use the Quartus software to program the FPGA.

4 Powering the MaxProLogic

You can run the MaxProLogic from a laptop with 2.5W of power. Or you can run it from the +5V @ 2A wall USB chargers for 10W of power. The barrel connector can handle up to +9V @ 3 A for 27W of power.

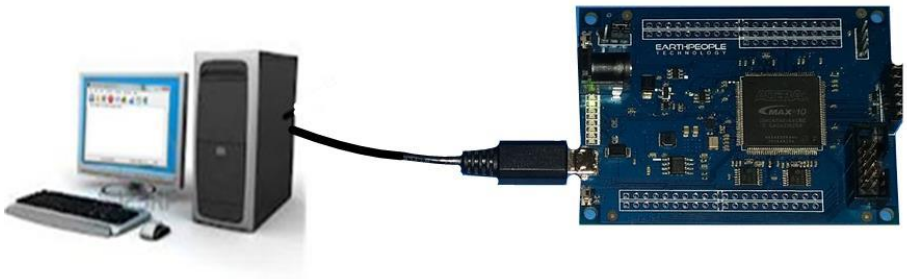
- Standard USB cable from Laptop/PC.
- +5 VDC wall charger (phone charger) through USB cable.
- +5.5 to +9 VDC supplied through the DC power jack.
- +5.5 to +9 VDC supplied through I/O connector pin.



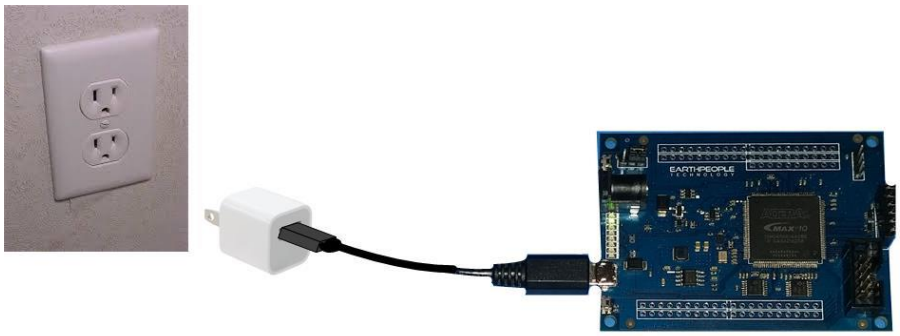
The barrel connector is the typical size used on many popular DIY boards such as the Arduino series. It has the following mechanical specs:

- 2.0mm Inner Diameter
- 5.5mm Outer Diameter

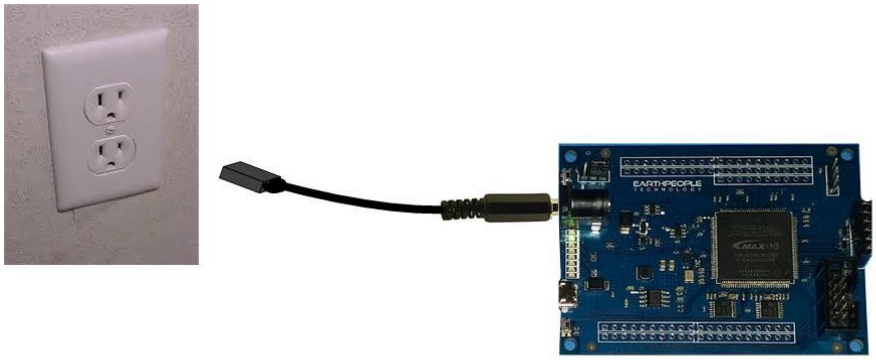
The barrel connector does not include a diode protection to prevent reverse polarity connection. So, care must be exercised when connecting up your cable to the barrel connector. Please ensure the correct polarity connections are made before connecting to the MaxProLogic. Also, there is no discrete protection to the power input. The power supply does include a high current protection circuit. The current limit is around 4.7Amps. But, the MaxProLogic is only designed to handle 2Amps of current. So, damage may occur to the MaxProLogic if the user does not exercise care in design and use of the Inputs/Outputs.



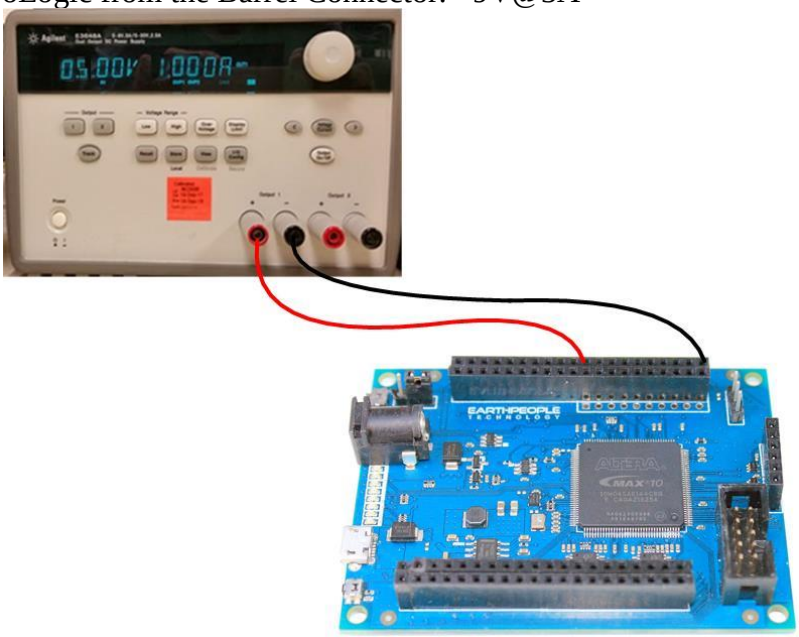
Power the MaxProLogic directly from the PC. +5V@0.5A



Power the MaxProLogic directly from the wall charger. +5V@2A



Power the MaxProLogic from the Barrel Connector. +9V@3A

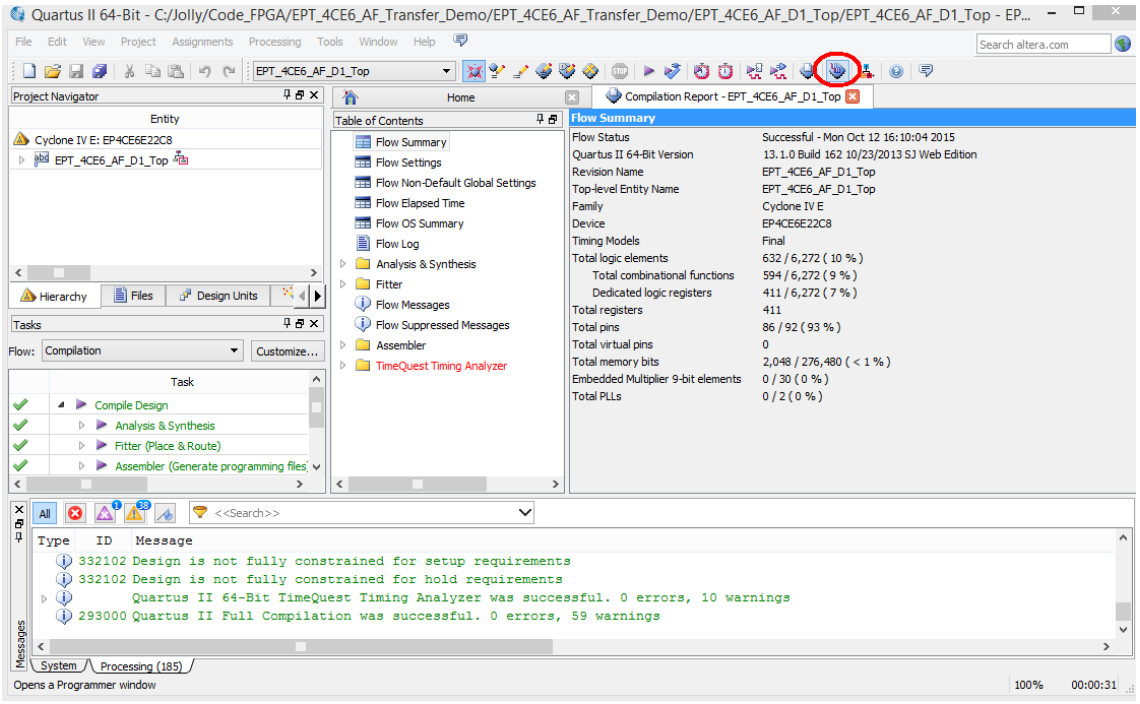


Power the MaxProLogic directly from a power supply. +5V@2A

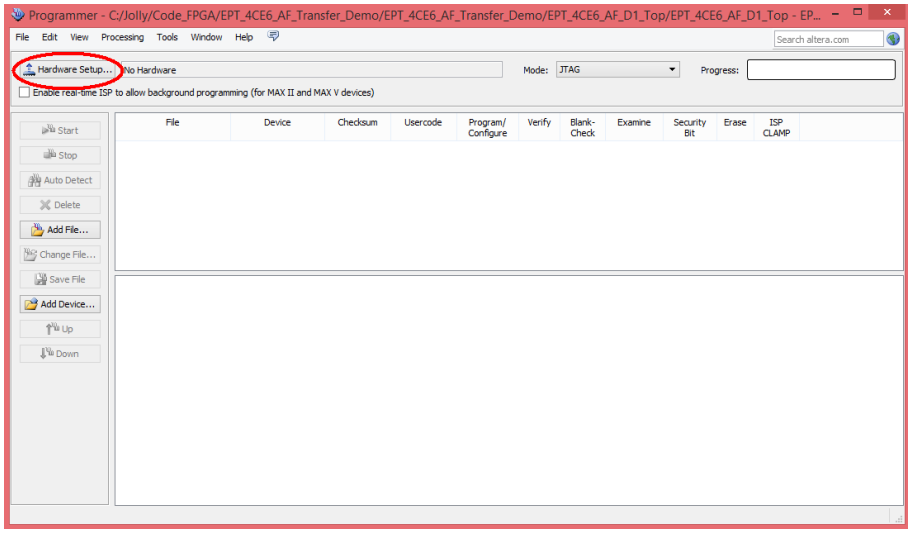
5 Programming the MaxProLogic

Configuring the FPGA is quick and easy. All that is required is a standard USB Micro B cable and a compatible Blaster Programmer. Connect the MaxProLogic to the PC, open up Quartus II, open the programmer tool, and click the Start button. To program the MAX10 Configuration Flash, connect the Blaster and ensure the JTAG Driver is loaded for Quartus II.

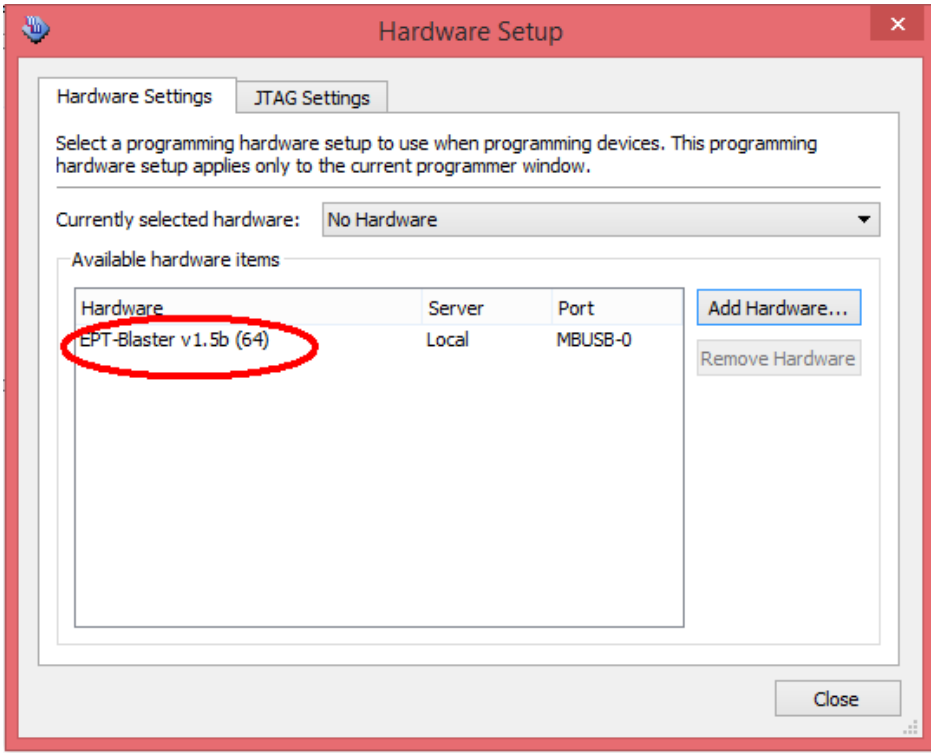
If the project created in the previous sections is not open, open it. Click on the Programmer button.



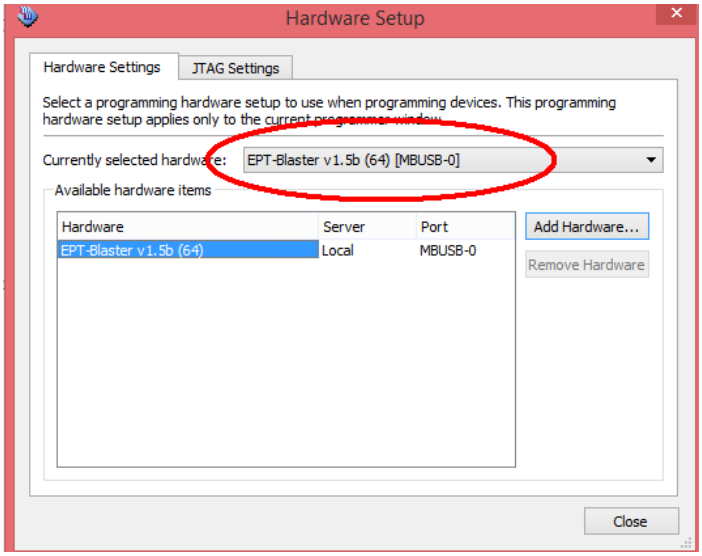
The Programmer Window will open up with the programming file selected. Click on the Hardware Setup button in the upper left corner.



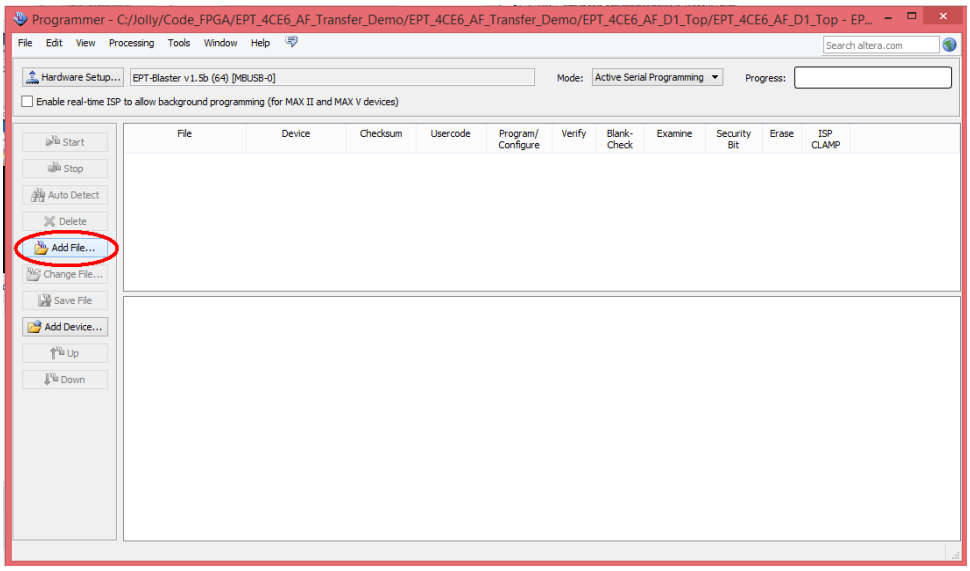
The Hardware Setup Window will open. In the “Available hardware items”, double click on “EPT-Blaster v1.6b”.

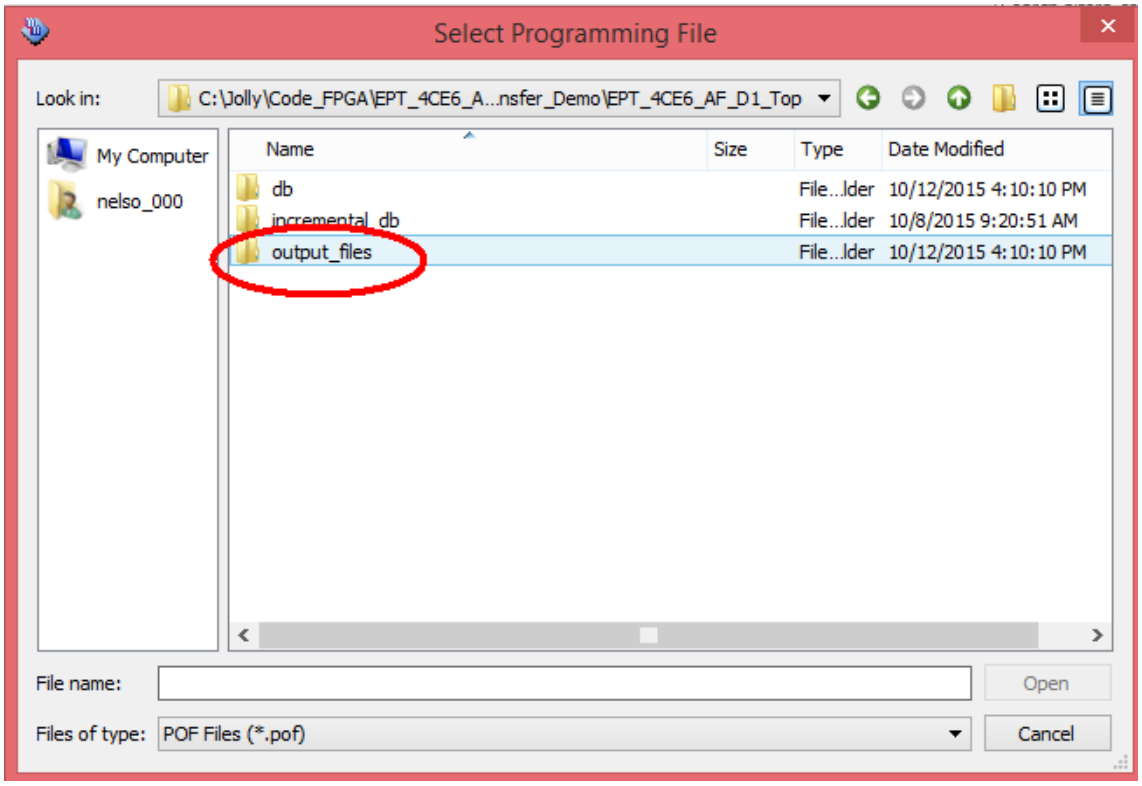


If you successfully double clicked, the “Currently selected hardware:” dropdown box will show the “EPT-Blaster v1.6b”.

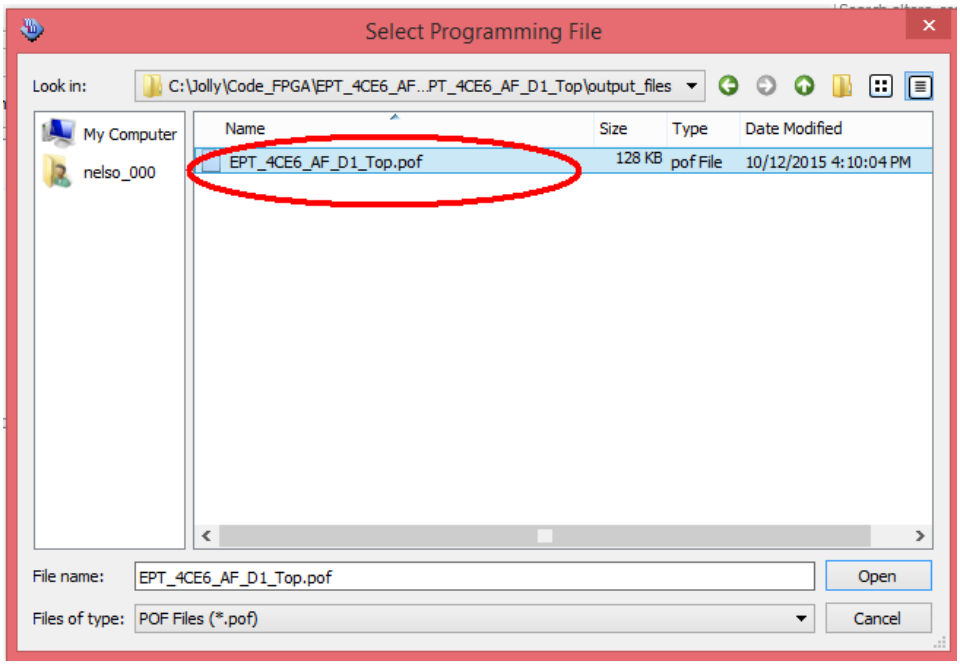


Click on the “Add File” button

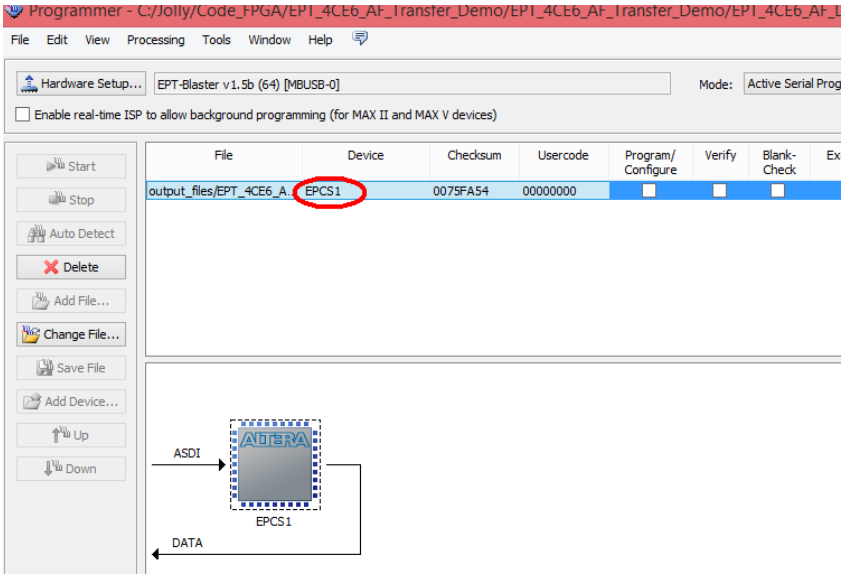




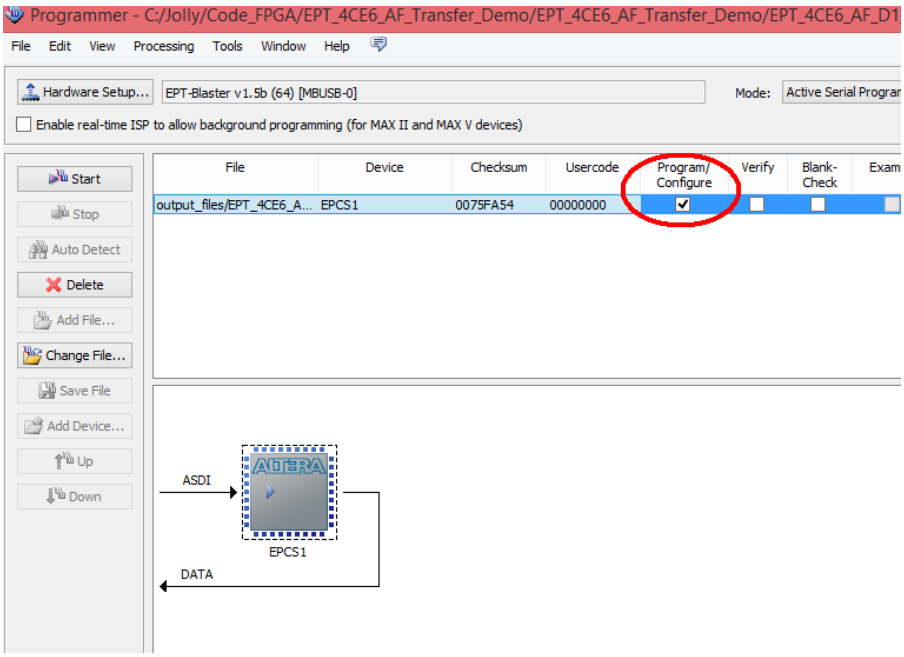
At the Browse window, double click on the output files folder.



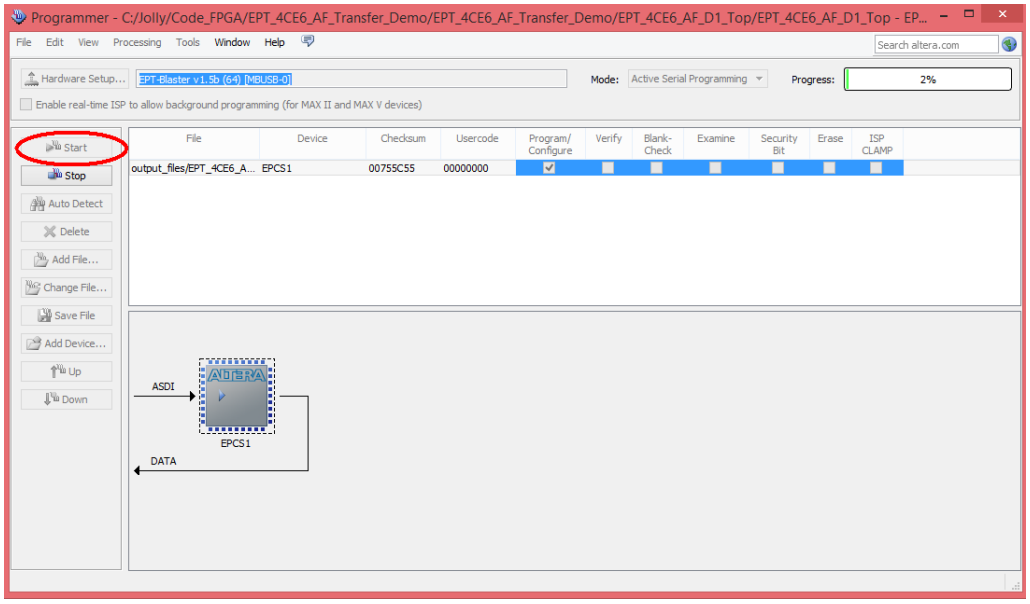
Double click on the "EPT_10M04_AF_Top.pof" file. Click the Open button in the lower right corner.



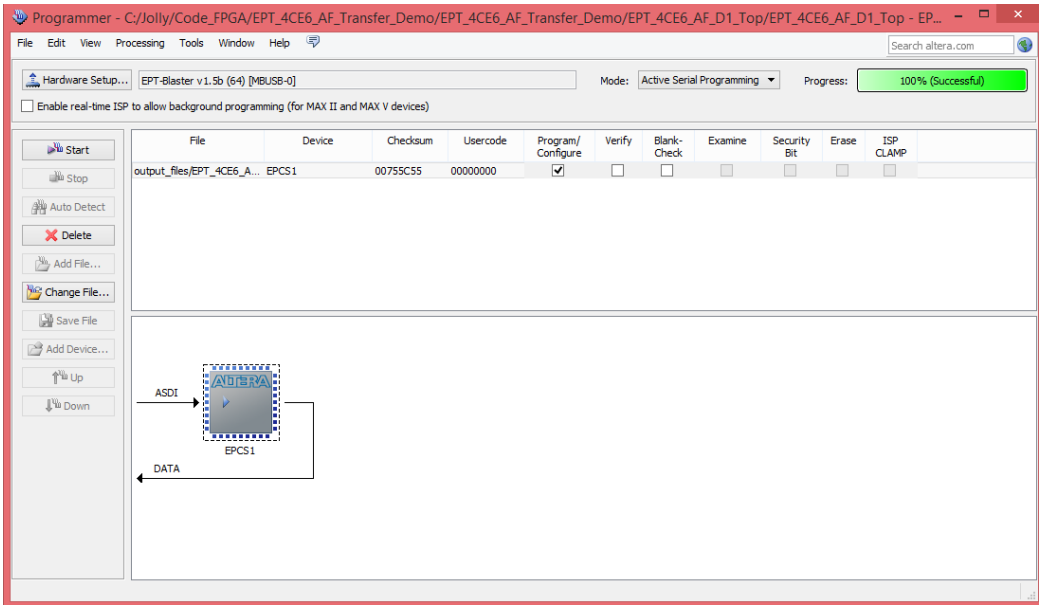
Next, selet the checkbox under the “Program/Configure” of the Programmer Tool.



Click on the Start button to to start programming the FPGA. The Progress bar will indicate the progress of programming.



When the programming is complete, the Progress bar will indicate success.



At this point, the MAXPROLOGIC is programmed and ready for use.